UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

ANÁLISIS Y DISEÑO DE LA METODOLOGÍA DE PROCESAMIENTO DE DATOS DISPONIBLES EN REDES IoT (INTERNET DE LAS COSAS) PARA IMPLEMENTAR LA GESTIÓN DE ALARMAS COMUNITARIAS EN LA CIUDAD DE POPAYÁN (COLOMBIA)

CARLOTA TARAZONA LIZARRAGA

2017

TRABAJO FIN DE GRADO

Título**:** **Análisis y diseño de la metodología de procesamiento de datos disponibles en redes IoT (Internet de las Cosas) para implementar la gestión de alarmas comunitarias en la ciudad de Popayán (Colombia)**

Autor**:** **Dª. Carlota Tarazona Lizarraga**

Tutor**:** **D. Manuel Lambea Olgado**

TRIBUNAL:

Presidente**:** **D.**

Vocal: **D.**

Secretario: **D.**

Suplente: **D.**

Fecha de lectura:

Calificación:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

ANÁLISIS Y DISEÑO DE LA METODOLOGÍA DE PROCESAMIENTO DE DATOS DISPONIBLES EN REDES IoT (INTERNET DE LAS COSAS) PARA IMPLEMENTAR LA GESTIÓN DE ALARMAS COMUNITARIAS EN LA CIUDAD DE POPAYÁN (COLOMBIA)

CARLOTA TARAZONA LIZARRAGA

2017

Resumen

Teniendo en cuenta los graves problemas de seguridad en América Latina, y en concreto en ciudades intermedias como Popayán (Colombia), están surgiendo proyectos de emprendedores de carácter tecnológico con el fin de mejorar los índices en la materia.

Este Trabajo de Fin de Grado surge de una iniciativa de colaboración con la Corporación Cluster Cratic, fundación sin ánimo de lucro, que apoya a emprendedores locales en la región del Cauca (Colombia), con el objetivo de apoyar un sistema de registro de alertas, desarrollado por uno de sus emprendimientos, llamado Hutek. Dicho sistema consiste en un chat a través de una aplicación móvil, en el que los usuarios pueden generar en tiempo real distintas alertas (robos, incendios, accidentes, etc).

Actualmente, los datos únicamente son generados, por lo que se persigue el objetivo de dotar a dichos datos de un valor añadido, de manera que se perfeccione el sistema ya implantado y se reduzca la inseguridad gracias a la tecnología. Para ello se han seguido tres líneas principales de trabajo:

- Almacenar los datos recolectados.
- Analizarlos y desarrollar un sistema de localización, que permita que se enciendan alarmas físicas distribuidas a lo largo de la ciudad.
- Implantar un sistema de visualización que facilite la toma de decisiones por parte de los actores clave.

Así mismo, se pretende que este trabajo contribuya al logro de tres de los Objetivos de Desarrollo Sostenible definidos por la ONU. Ya que este tipo de iniciativas pueden impactar muy positivamente en la vida de los ciudadanos de Popayán, y favorecer su transición en una ciudad inteligente.

Para lograr el objetivo planteado, en primer lugar, se ha realizado un análisis de las diferentes tecnologías para poder abordar las líneas de trabajo mencionadas, tras ello se ha definido la arquitectura de software desde diferentes tipos de modelos, se ha desarrollado e implementado el sistema y por último se han realizado unas pruebas que valoren los resultados obtenidos.

En este documento se expone el proceso seguido para contribuir a que se reduzca la inseguridad en la ciudad de Popayán.

Palabras clave

Bot, Seguridad, Alarmas, Visualización, Almacenamiento, Localización

Summary

Considering Latin America's serious security issues, particularly in middle cities like Popayan (Colombia) several technological entrepreneurships are emerging with the specific goal to decrease crime rates.

This work is based on a collaboration initiative with the Cluster Creatic Corporation, a non-profit foundation that support local entrepreneurs from Cauca region (Colombia) contributing to the development of an alarm registry system created by one of those entrepreneurship, called *Hutek*. That system is based on a mobile application chat where the users can generate real time alerts (thefts, fires, accidents, emergencies, etc.).

Currently, data is only generated, and therefore the aim is to endow data with added value. In that way, the implemented system may be improved and the crime rates can be decreased through technology. To achieve that, the three principal work lines are:

- Storing the collected data.
- Analyzing the collected data and developing a location system that allows triggering physical alarms distributed around the city.
- Implementing a visualization system that enables the decision-making process by the key stakeholders.

Moreover, this work seeks to contribute to the accomplishment of the UN Sustainable Development Objectives because this type of initiatives may impact positively on Popayan citizens' daily life and favor their transition to become a smart city.

To achieve this goal, in first place an analysis has been made about different technologies to develop the traced work lines. After that, a software architecture has been defined from different kind of models, and the system has been developed and implemented. Finally, several tests have been undertaken to value the obtained results.

In this document is exposed the process followed to contribute in the crime rate reduction in Popayan city.

Keywords

Bot, Security, Alarms, Visualization, Storage, Location

GLOSARIO

AJAX: Asynchronous JavaScript And XML, es una técnica de desarrollo web para crear aplicaciones interactivas.

API: Application Programming Interface, es una interfaz de programación de aplicaciones, un conjunto de subrutinas, funciones y procedimientos que ofrece ciertas bibliotecas para ser utilizado por otro software como una capa de abstracción.

CSS: Cascading Style Sheets (hojas de estilo en cascada), es el lenguaje utilizado para describir la presentación de documentos HTML.

CSS3: es la última evolución del lenguaje CSS.

GitHub: es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.

HTML: HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web.

HTML5: es la quinta revisión del lenguaje HTML de la World Wide Web.

HTTP: HyperText Transfer Protocol, es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

JSON: JavaScript Object Notation, es un formato de texto ligero empleado para el intercambio de datos.

MVC: Modelo-Vista-Controlador, patrón de arquitectura de software basado en la reutilización de código y la separación de conceptos para facilitar la tarea al desarrollador.

UML: Unified Modeling Language, es un lenguaje de modelado de sistemas de software. Se utiliza para visualizar, especificar, definir y documentar un sistema software de forma gráfica.

URL: Uniform Resource Locator, se trata de la secuencia de caracteres que sigue un estándar y que permite denominar recursos dentro del entorno de Internet para que puedan ser localizados.

SMTP: Simple Mail Trasnfer Protocol, es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico entre ordenadores y otros dispositivos.

Stackoverflow: sitio web utilizado por una comunidad de desarrolladores informáticos, en la cual otros desarrolladores pueden encontrar soluciones a problemas de programación en diferentes lenguajes.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. CONTEXTO

Este trabajo surge de una iniciativa de colaboración con la corporación Cluster Creatic. Esta organización es una entidad cuyas actividades se enfocan en el desarrollo tecnológico e innovación aplicada en conjunto con el sector empresarial de Colombia específicamente en la región del Cauca a través del apoyo a emprendimientos de base tecnológica. Esta entidad es una fundación sin ánimo de lucro, ya que se apoya en proyectos del estado. Como estrategia de colaboración interinstitucional la UPM ha realizado trabajos en colaboración con este tipo de instituciones alrededor del mundo. A raíz de esto, este trabajo se ha desarrollado en la localización de la corporación Cluster creatic, la ciudad de Popayán.

América Latina ha reportado un crecimiento económico significativo y sus índices de superación de la pobreza, sanidad y educación han mejorado notoriamente [1]. Sin embargo, la violencia e inseguridad, especialmente en las ciudades, ha empeorado con graves consecuencias a nivel económico, político y social para las naciones latinoamericanas.

Además, según el Foro sobre Desarrollo de América Latina del Banco Mundial, se trata de la región *"más violenta del mundo, con un promedio de 24 homicidios por cada 100.000 habitantes"*, y *"representa solo el 8 % de la población mundial, pero el 37 % de los homicidios en el mundo ocurren en ella*". La tasa de homicidios, de hecho, se aceleró durante la segunda mitad de la década. Cada 15 minutos, al menos cuatro personas son víctimas de homicidio en América Latina y el Caribe. En 2013, de las 50 ciudades más violentas del mundo, 42 se encontraban en la región.

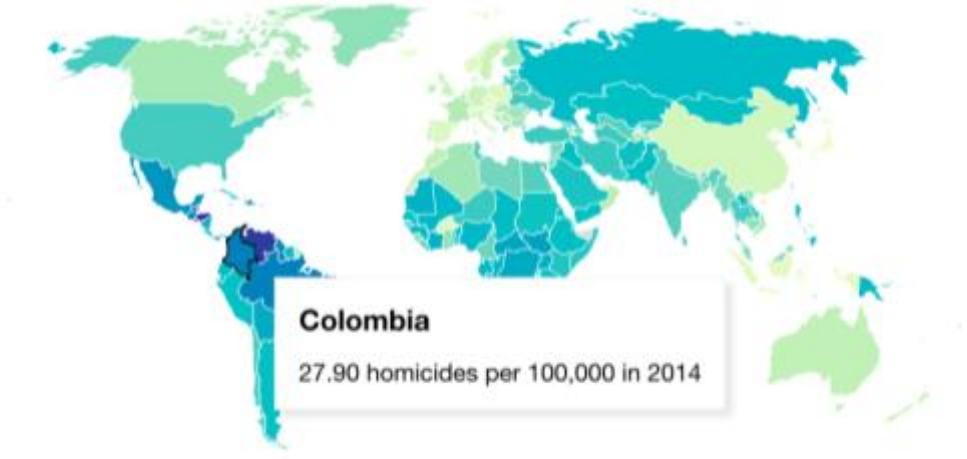

Figura 1 Mapa Mundial- Homicidios por cada 100.000 habitantes

Como se puede observar en la *Figura 1,* que representa los datos recogidos por el Banco Mundial, en Colombia se produjeron en 2014 27.9 homicidios por cada 100.000 habitantes. En contraste con España, donde se produjeron 0.7 por cada 100.000 habitantes.

No es de extrañar que el número de latinoamericanos que mencionan al delito como su mayor preocupación se haya triplicado en estos años. "*La violencia hace que las personas se retraigan, se oculten detrás de sus puertas y eviten los espacios públicos, debilitando los lazos interpersonales y sociales que nos unen como comunidad*" [2]

Concretamente, Colombia es uno de los países de la región donde más alto ha sido el coste de la violencia. En concreto, algunas de las ciudades intermedias en el país han sido identificadas como focos de especial atención en materia de seguridad y convivencia. Popayán, la capital del departamento del Cauca es considerada como ciudad intermedia por tener más de 100.000 habitantes (277.540 habitantes). En materia de seguridad, es una ciudad con una alta tasa de hurtos (750 por cada 100.000 habitantes), homicidios (31 homicidios por cada 100.000 habitantes) y lesiones (400 por cada 100.000 habitantes), según el Plan de Desarrollo Municipal 2016-2019 de la actual administración municipal. Igualmente, conforme a un estudio realizado por el Departamento Administrativo Nacional de Estadística (DANE) es una de las ciudades con mayor tasa de victimización (que sus habitantes han sido víctimas de hurto a residencia, hurto a personas, hurto a vehículo, involucramiento en riñas y peleas que impliquen violencia física, y extorsión o intento de extorsión) del país. En este ranking se posiciona como la cuarta ciudad de Colombia con mayor tasa de victimización.

Actualmente en la ciudad de Popayán se están generando soluciones tecnológicas para este tipo de problemas. En concreto, para reducir esta inseguridad la empresa perteneciente a la organización con la que se colabora en este trabajo, Hutek, ha implantado un sistema de registro de datos. Este sistema utiliza un chat a través de una aplicación móvil (Telegram), que permite generar en tiempo real alertas de diferente índole (violencia, incendios, etc.), comunicándose con bots (asistentes virtuales). Estos son un tipo de programa informático autónomo capaz de llevar a cabo tareas concretas e imitar el comportamiento humano.

1.1 PLANTEAMIENTO DEL PROBLEMA

Visto el trabajo de empresas como Hutek que tratan la reducción de índices de inseguridad en la ciudad a través de la tecnología, se hace evidente la necesidad de apoyar este tipo de iniciativas fortaleciendo el sistema a través de procesos de sofisticación o mejora que permitan una mayor interacción por parte de usuarios.

Una de las principales debilidades que tiene actualmente el sistema desarrollado es que carece de interfaces amigables al usuario, siendo la visualización de información nula para diferentes actores. Esto genera impactos negativos ya que no se puede usar la información recolectada de las alarmas generadas por parte de usuarios estratégicos como policía, alcaldías, bomberos, etc, para mejorar las condiciones y tomar acciones a partir de decisiones que puedan ser soportadas por el sistema.

Es por esto que el problema a solucionar por este TFG es el de agregar un valor añadido al sistema, de forma que los usuarios puedan sentirse más seguros y cooperar como comunidad para lograrlo

1.2 OBJETIVOS

Una vez expuesto el contexto y el planteamiento del problema, resulta necesario llegar al objetivo de reducir la inseguridad de los ciudadanos de Popayán, Colombia. Para, de esta forma, reducir la brecha que separa al país del cumplimiento efectivo de los objetivos de desarrollo sostenible establecidos por la Organización de Naciones Unidas y 193 líderes mundiales en 2015. De los 17 objetivos planteados, el trabajo hace especial énfasis en el cumplimiento de tres de ellos: 3. Garantizar una vida sana y promover el bienestar para todos en todas las edades; 11. Lograr que las ciudades y los asentamientos humanos sean inclusivos, seguros, resilientes y sostenibles; y, 16. Promover sociedades justas, pacíficas e inclusivas.

A partir de esto, se describen a continuación los objetivos de este TFG.

Objetivo general

Apoyar en la construcción de un sistema que contribuya a mejorar la seguridad de los ciudadanos de Popayán

Objetivos específicos:

- Diseñar e implantar un sistema de almacenamiento de los datos generados en la comunicación usuario-bot *PopayanAmigo*.

- Desarrollar un algoritmo que permita localizar el sector o cuadrante más cercano al que se encuentra el usuario que ha generado la alerta a partir de su ubicación, obtenida de la comunicación entre usuario y bot; y que así puedan sonar una serie de alarmas físicas distribuidas en la ciudad e informar a los servicios de seguridad que se encuentran más próximos

- Implantar un prototipo que permita visualizar los datos almacenados, obteniendo diferentes estadísticas, para que sean analizados y sirvan para futuras tomas de decisiones a diferentes actores: Policía y Ayuntamiento de la ciudad de Popayán.

2 ESTADO DEL ARTE

En esta sección se va a introducir al lector para contextualizarle en el marco tecnológico en el que se encuentra el trabajo. Para ello se presentan las tendencias tecnológicas que se pueden utilizar para abordar el problema planteado. Así como las tecnologías relacionadas con el presente proyecto, presentando posteriormente las tecnologías concretas empleadas para abordar cada objetivo del trabajo.

2.1 TENDENCIAS

Las tendencias tecnológicas manejadas en el proyecto corresponden a: Smart Cities, Internet de las Cosas, Bots, Computación en la nube y Almacenamiento y Visualización de datos. A continuación, se realiza una pequeña descripción de cada una y como se enmarcan en este trabajo.

2.1.1 Smart Cities

Una ciudad inteligente es una ciudad sostenible que hace un uso intensivo de las tecnologías de la información y la comunicación para garantizar la buena planificación y gestión de sus recursos, con el fin de incrementar la calidad de vida de sus habitantes. En este sentido cabe resaltar los objetivos de desarrollo sostenible identificados por la ONU y alineados con la OCDE:

Se resalta que esta propuesta hace especial énfasis en el objetivo 3: Salud y bienestar y el objetivo 11: ciudades y comunidades sostenibles, como caso piloto de ciudad inteligente para

la región. En este sentido, se considera la necesidad de hacer las ciudades más seguras y cómo contribuir como comunidad para lograrlo.
En este orden de ideas, es necesario definir mecanismos que permitan integrar las tecnologías para la transición de las ciudades a ciudades inteligentes, tecnologías como: internet de las cosas, análisis y procesamiento de datos y computación en la nube, entre otras.

En Popayán se están desarrollando iniciativas para que sea una ciudad más inteligente. En concreto, este sistema ayuda a mejorar la seguridad de sus habitantes y que cooperen como comunidad para conseguirlo, y lo logren gracias a sus dispositivos móviles conectados a Internet.

2.1.2 Internet de las Cosas (o Internet of Things - IoT)

El concepto de IoT hace referencia a la interconexión de elementos cotidianos entre si y a través de Internet. Esto produce una generación de datos de diferentes fuentes en tiempo real a una velocidad que crece exponencialmente. Esta situación está provocando que la comunicación adopte una nueva dimensión, permitiendo nuevos usos y aplicaciones entre los dispositivos de uso diario y sus usuarios, para mejorar la experiencia de éstos.

Esta tecnología puede incluso transformar la forma en la que las ciudades se construyen y sus habitantes interactúan con ellas. Además, puede afectar positivamente en la manera en la que los ciudadanos cooperan entre sí generando una comunidad activa y solidaria. El concepto de ciudades inteligentes (o *smart cities)* sería casi imposible de entender sin la tecnología IoT, ya que ésta consigue emplear los recursos de forma más eficiente y ofrecer servicios a los ciudadanos para que mejoren su calidad de vida.

Según un informe del CISCO en el año 2020 habrá más de 11.600 millones de dispositivos móviles conectados, por lo que es vital implementar sistemas que sigan estas tendencias, ofreciendo servicios que solventen las necesidades de los habitantes de las ciudades, como en este caso resolver problemas de seguridad urbana.

En este trabajo la tendencia IoT se representa en cómo unos módulos recolectan los datos, otros los almacenan y activan alarmas físicas. Todo ello se consigue gracias a la conexión tanto de los dispositivos móviles de los usuarios, como de las alarmas físicas distribuidas por la ciudad, a Internet.

2.1.3 BOTS

Empleado como aféresis de robot, su definición estándar (un programa o agente informático autónomo, diseñado para imitar el comportamiento de usuarios humanos en tareas concretas y repetitivas) no proporciona una idea muy precisa de la variedad de aplicaciones con las que pueden asociarse. [3]

Se pueden desarrollar en cualquier lenguaje de programación. Suelen estar configurados para funcionar en redes, especialmente en Internet, e interactuar con otros sistemas o usuarios en tareas como la edición de textos, moderar conversaciones, responder a preguntas frecuentes –sobre un servicio, recurso informático o búsqueda de contenidos en la web, como hacen los bots conversacionales–, enviar correos electrónicos o dinamizar videojuegos, entre otras. [4]

Como se ha comentado en el contexto, la empresa Hutek de la ciudad de Popayán ya tiene un sistema desarrollado. Han implementado un bot en la aplicación móvil Telegram llamado *PopayanAmigo* mediante el cual los habitantes de la ciudad pueden publicar alertas de diferente índole. Para ello los usuarios establecen una conversación con el mismo introduciendo sus datos, y se les proporciona un menú para seleccionar el tipo de alarma que les ha sucedido, incluyendo también su ubicación. Este bot se podría considerar conversacional, ya que mantiene una conversación con los usuarios imitando el comportamiento humano.

Por otro lado, han implementado diversos bots distribuidos por la ciudad encargados de la comunicación con la nube y con la API desarrollada. A lo largo de esta memoria se llamará a estos bots *Bots Físicos*, ya que se encuentran asociados a alarmas físicas.

2.1.4 Computación en la nube (o Cloud Computing)

La computación en la nube según el Instituto Nacional de Normas y Tecnología NIST (National Institute of Standards and Technology), "*es un modo de estructurar y usar las TIC, que permite el acceso ubicuo y adaptable a un conjunto compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente aprovisionados y liberados con un esfuerzo mínimo de gestión o interacción mínima por parte del proveedor de servicios*".

Gartner, empresa consultora y de investigación de las tecnologías de la información define la computación en la nube como *"un estilo de computación donde se ofrecen, de forma escalable y elástica, capacidades TI (Tecnologías de la Información), utilizando Internet".*

El concepto de Computación en la nube, que se aplica en este proyecto conjunto, radica en la posibilidad de alojar las infraestructuras, servicios o aplicaciones en la nube.

En este caso se utiliza el servicio en la nube ofrecido por Microsoft, conocido como Azure[1], para alojar todo el sistema desarrollado, es decir, la API, el sistema de almacenamiento y visualización de datos, etc. en sus centros de datos.

[1] Microsoft Azure: https://azure.microsoft.com/es-es/

2.1.5 Almacenamiento de datos

El primer paso, previo a la carga de la información en el almacén de datos, es la integración de estos. El proceso que se debe seguir es el siguiente:

1. Extracción o recolección de los datos de las fuentes correspondientes.
2. Transformación de los datos (operaciones de depuración, consolidación, resumen y reestructuración de datos).
3. Cargar en el ámbito de almacenamiento de datos.

A este conjunto de operaciones se las conoce como ETL (Extract - Transform - Load). [5]

En este trabajo, el almacenamiento de los datos se implementa a través de una única base de datos y los procesos de recolección de los mismos son realizados mediante las interfaces proveídas en el sistema.

2.1.6 Visualización de datos

Las herramientas de visualización de datos son tecnologías que permiten la visualización y, en ocasiones, la interpretación de los datos. En este conjunto se incluyen imágenes digitales, sistemas de información geográficos, interfaces visuales, gráficos, realidad virtual, presentaciones dimensionales, vídeos y animación. Estas herramientas ayudan a los diferentes actores a tomar decisiones y llevar a cabo diferentes planes o políticas en base a esas decisiones; se pueden implementar en cualquier ámbito.

En este caso la representación de la información para la visualización por parte de los actores que accederán a ella (Policía y Ayuntamiento de Popayán) se lleva a cabo tomando los datos recolectados y almacenados y mostrándolos en un conjunto de gráficas sencillas e intuitivas. El objetivo es permitirles visualizar la información y tomar decisiones para implementar políticas que mejoren la seguridad de la ciudad.

2.2 SELECCIÓN DE TECNOLOGIAS EMPLEADAS

En este apartado se va a especificar cómo se llevó a cabo la selección de las diferentes tecnologías empleadas en la implementación del sistema y, a su vez, se va a realizar una breve descripción de cada una de ellas.

2.2.1 Frameworks

En el desarrollo de aplicaciones web se pueden encontrar una amplia variedad de herramientas con determinadas características que reducen el trabajo, facilitando la construcción de las aplicaciones. Estas herramientas se denominan *frameworks* y su uso permite que los desarrolladores consuman menos tiempo y recursos en la construcción de las mismas. Los términos *front end* y *back end*, empleados para llamar a los *frameworks*, sirven para separar una capa de presentación de una capa de acceso a datos, respectivamente.

Para escoger los *frameworks* a utilizar tanto *back end* como *front end*, se llevó a cabo el siguiente proceso:

1. Se hizo una primera selección de tecnologías basada en la experiencia del equipo, considerando que estas fueran óptimas para el desarrollo del prototipo de este TFG.
2. Se establecieron unos criterios de selección.
3. Se evaluaron esos criterios y se tomó una decisión.

A continuación, se van a adjuntar dos tablas comparativas para la elección de los *frameworks back end* y *fron end*. Los criterios seguidos son:

1. Soporte de la comunidad (columnas 1 y 2): se ha valorado en base al número de consultas en *stackoverflow* y al número de repositorios en *GitHub*.

2. Soporte para el procesamiento de datos: valorado según *The Data Science,* que expone que los lenguajes de programación más populares para el análisis de datos son: Python, R, SAS, Matlab y Scala. [3]

3. Experiencia de uso de los integrantes del proyecto: valorado según una encuesta realizada a los desarrolladores tanto de la organización Cluster Creatic como a los emprendedores que forman Hutek.

	Soporte 1 (nº consultas stack)	Soporte 2 (nº repositorios en git)	Soporte para el procesamiento de datos	Soporte para aplicaciones web (cualitativo)	Experiencia de uso de los integrantes del proyecto [1-5]
Python (Django)	142.1K	99K	5	5	3.5
Ruby (Ruby on Rails)	272.5K	222K	2	5	1.6
JavaScript (Nodejs)	170K	91K	2	5	4.2

Tabla 1: Comparación framework backend

	Soporte 1 (nº consultas en stackoverflaw)	Soporte 2 (nº repositorios en github)	Integración con Django (nº repositorios públicos)	Experiencia de uso de los integrantes del proyecto [1-5]
Materialize	1K	2K	39	4
Bootstrap	17.4K	91K	1244	3.6
Foundation	4.4K	16K	81	1

Tabla 2: Comparación framework frontend

De acuerdo a la *Tabla 1*, se optó por Django, ya que, a pesar de no tener el soporte más alto de la comunidad, los integrantes del proyecto tienen una gran experiencia de uso y entre las tres opciones es el más valorado para el procesamiento de datos (criterio 2).

Tras la inclinación por Django, se evaluaron los *frameworks front end*, teniendo en cuenta el criterio de integración con Django. Por lo que se escogió Bootstrap principalmente debido a su gran soporte de la comunidad y por ser el que tiene mayor integración con Django. Como se puede observar en la *Tabla 2*.

Tras la selección de cada uno de ellos se hace una descripción de los mismos:

Framework back end: Django

Django utiliza una adaptación del patrón de arquitectura de software Modelo-Vista-Controlador (MVC) tradicional basado en la reutilización de código y la separación de conceptos, para facilitar la tarea al desarrollador. Esta adaptación empleada por Django se conoce como *Model-Template-View*. [6]

El componente **Template** define la interfaz gráfica, es decir, presenta la información contenida en el modelo de la forma indicada en la lógica (View). Uno de los objetivos definidos en este trabajo es la visualización de los datos por parte de los diferentes actores (Policía, Ayuntamiento, etc.) para futuras tomas de decisiones. Por lo tanto, tiene especial relevancia diseñar una página web sencilla, intuitiva y descriptiva que muestre los datos de la mejor manera posible.

Django genera HTML de manera dinámica, ya que es un *framework* web, y se encarga de esto a través de los templates o plantillas. En el caso de que no se implementen las plantillas, Django provee su propia interfaz gráfica para mostrar la información almacenada en la API. En cambio, para desarrollar la interfaz gráfica es necesario hacerlo en este módulo.

El componente **View** se corresponde con el controlador en una arquitectura MVC. Django tiene el concepto de «vistas» para encapsular la lógica responsable de procesar una petición del usuario y devolver la respuesta, es decir, responden a los eventos que realiza el usuario

sobre la interfaz e invocan al modelo para obtener información cuando es necesario, después devuelven esta información al componente Template. Por lo que se pueden considerar intermediarias entre el modelo y template. Son métodos que reciben una petición web y devuelven una respuesta HTTP que contiene la información dinámica necesaria para generar el HTML definido en el módulo Template.

El componente **Model** es la fuente única de información sobre los datos. Contiene los campos y el comportamiento de los datos almacenados. Generalmente, cada modelo se asigna a una tabla de una base de datos única. Es la parte encargada de estructurar los datos.

Este sistema se utilizó para hacer la parte lógica de la aplicación, recolección de los datos, y el posterior almacenamiento en la base de datos. Sus componentes, previamente descritos, han sido desarrollados durante la construcción del sistema.

Framework front end: Bootstrap

Como respuesta a la gran variedad de dispositivos conectados a Internet actualmente, nace el deseo de poder brindar una buena experiencia de usuario y que sea independiente del dispositivo que se esté empleando en la navegación. Por este motivo nace el concepto de "*Responsive Design"*. Su idea principal es evitar tener que realizar un diseño para cada uno de los dispositivos que se vayan a conectar, sino por el contrario buscar la flexibilidad, haciendo un único diseño que sea capaz de adaptarse a las necesidades de cada dispositivo, siendo invisible dicha transformación y consiguiendo que la usabilidad de la aplicación sea cómoda y atractiva para el cliente.

Bootstrap, desarrollado por Twitter, es un *framework* libre para la presentación y estilo de páginas web usando CSS3 y HTML5. Aporta una serie de elementos visuales y reglas que facilitan el diseño de un sitio web *Responsive.* Asimismo, ofrece varias plantillas predefinidas que permiten aplicar varios estilos visuales a la página. En el caso de este TFG será necesario encontrar una plantilla que se adapte al estilo de visualización de datos.

Por otro lado, es necesario destacar la gran integración que tiene Bootstrap con los *frameworks back end* como Django, Ruby on Rails. Por lo que ha sido un aspecto clave a la hora de seleccionarlo entre los distintos *frameworks front end*, su buena integración con el previamente elegido Django.

Este *framework* se ha empleado para conectarse a la parte lógica y mostrar los datos a los usuarios, mediante herramientas de visualización (gráficas), que se expondrán más adelante.

2.2.2 Base de datos

Una base de datos se define como una colección, generalmente grande, de datos organizada de una manera determinada que facilite su búsqueda y recuperación por parte de un ordenador. [7]

Las bases de datos se pueden dividir en dos grandes grupos, SQL y NoSQL. En la tabla 3 se realiza una comparación entre ellas en base a las siguientes características: modelo, tipo de datos, esquema, transacciones y rendimiento.

	No SQL	SQL
Modelo	No relacional Almacena los datos en documentos JSON, columnas anchas, pares clave-valor	Relacional Almacena los datos en tablas
Datos	Ofrece flexibilidad, no todos los registros deben ser del mismo tipo Se pueden añadir propiedades sobre la marcha Datos semiestructurados o complejos	Todos los registros deben tener las mismas propiedades Para añadir nuevas propiedades es necesario modificar los esquemas de datos Datos estructurados
Esquema	Esquema dinámico o flexible La base de datos es independiente del esquema, y éste depende de la aplicación	Esquema fijo El esquema debe estar sincronizado entre la base de datos y la aplicación
Transacciones	El soporte de transacciones ACID[2] varía según la solución	Soporta totalmente transacciones ACID
Rendimiento	Si es necesario, el rendimiento se puede aumentar reduciendo la consistencia Toda la información sobre una entidad esta típicamente en un solo registro, por lo que una operación se puede actualizar	El rendimiento se puede maximizar mediante la ampliación de los recursos disponibles y el uso de estructuras en memoria La información sobre una entidad puede extenderse a través de muchas tablas y filas, requiriendo muchas combinaciones para completar y actualizar una consulta

Tabla 3: Comparación Bases de datos [8]

[2] En el caso de las bases de datos, el concepto de transacciones ACID se refiere a un conjunto de 4 propiedades, las cuales garantizan que las transacciones se realicen de forma confiable. Éstas características son: Atomicidad, Consistencia, Aislamiento ("Isolation" en inglés) y Durabilidad. Además, cabe destacar el significado de transacción en el ámbito de bases de datos, el cual se refiere a una única operación lógica, aunque la misma involucre pasar por diferentes tablas.

En el sistema implementado en este trabajo se considera una ventaja que los datos se encuentren estructurados y en un esquema fijo, ya que son muy concretos y se necesita seguir el esquema Entidad-Relación entre ellos. Además, es necesario que los datos sean estructurados, ya que permite un fácil acceso. Por lo que se ha seleccionado una base de datos de tipo relacional.

Dentro del conjunto de bases de datos relacionales: MySQL, SQLite, Oracle, PostgreSQL, Microsoft SQL Server, etc. Se ha seleccionado una de las más populares, y que además permite un posterior soporte, ésta es PostgreSQL. Sin embargo, destacando la posibilidad que ofrece Django (con Python) de migrar las bases de datos de una manera rápida y sencilla, la selección de la base de datos concreta no es algo que se considere relevante.

2.2.3 Entornos de despliegue del sistema

En el transcurso de la implementación de este sistema se ha trabajado en dos entornos. Se presentan a continuación:

- Entorno de Desarrollo: donde se ha desarrollado toda la aplicación, y se han ido realizando pruebas a medida que se progresaba. Este entorno de trabajo se corresponde con un ordenador de uso personal, cuyas características son: MacBook Air con Procesador 1,7 GHz Intel Core i5 y Memoria 4 GB 1333 MHz.

- Entorno de Pruebas: implementado en la nube de Azure, con una máquina virtual con las siguientes características:
  DNS name perfeccionamientoviewer.cloudapp.net
  Operating system Linux
  Size Standard D1(1 Core, 3.5 GB memory)
  En este entorno de trabajo se desplegó el sistema para realizar pruebas con usuarios reales.

3 ARQUITECTURA

La arquitectura de software de un sistema informático se define como el conjunto de estructuras necesarias para razonar sobre el mismo. Estas estructuras comprenden los elementos de software, las relaciones entre dichos elementos y sus propiedades. [9]

Es imprescindible documentarla de manera correcta para que otros posibles usuarios puedan mantener, o incluso construir otro sistema a partir de éste. En el caso concreto de este TFG, al tratarse de una colaboración con otra entidad (o desarrolladores) es necesario definir claramente todos los componentes y posibles funcionalidades del software desarrollado.

3.1 MODELOS Y DESCRIPCIÓN

Para desarrollar la arquitectura del presente trabajo se ha seguido como referencia el enfoque de "*Views and Beyond*" del libro *Documenting Software Arquitecture*, desarrollado por "*Software Engineering Institute*" (Carnegie Mellon University). [10]

Su planteamiento principal está enfocado en las vistas del sistema, y en cómo se documentan desde diferentes perspectivas. Una vista se puede definir como la representación de las múltiples estructuras de un sistema de software y las relaciones entre ellas.

Por ello, para el diseño de este trabajo se ha empleado el lenguaje de modelado estándar UML *(Unified Modeling Language)*, que visualiza, especifica y documenta cada una de las partes que comprende el desarrollo de software.

3.1.1 Modelo basado en capas

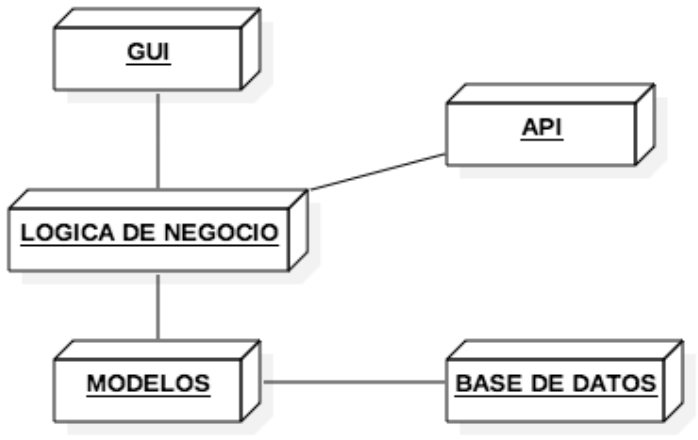

Figura 2: Diagrama de Capas

En el modelo basado en capas, cada capa representa una agrupación de módulos que ofrecen un conjunto cohesivo de servicios. Estos pueden ser cualquier componente, desde módulos que implementen un servicio web, hasta subrutinas de un lenguaje determinado. Uno de los requisitos fundamentales es que todas sus capas estén dotadas de una interfaz mediante la cual los servicios puedan ser accedidos o activados. También resulta imprescindible definir las relaciones entre ellas. [9]

La primera capa, *GUI* (Interfaz Gráfica de Usuario), permite la interacción del usuario con el programa de una manera visual. En este trabajo esta capa está formada por el módulo de las *templates* de Django (archivos .html) que conforman la aplicación web mediante la cual el usuario se conecta para visualizar la gestión de usuarios, gráficas de alarmas, mapas de ubicación de bots, entre otros.

La capa de *Lógica de Negocio* se encarga de encapsular la lógica responsable del procesado de peticiones del usuario y la devolución de respuestas al mismo, esta capa se relaciona con la *GUI* al mostrar al usuario la información generada dentro de los métodos que procesan las peticiones generadas por el usuario, que en la mayoría de los casos corresponde a información modelada por la capa del *Modelo*.

La capa del *API*, es el punto de comunicación usado por sistemas externos para acceder a información relevante del sistema. En este sistema el bot *PopayanAmigo* y los mencionados *Bots Físicos* utilizan esta capa para publicar y consultar información relevante sobre las alarmas y su activación. A diferencia de la capa *GUI*, mediante la que se conectan los usuarios que van a emplear el sitio web de visualización.

La capa *Modelos* se encarga de mapear la información en la base de datos. Se trata de una capa de abstracción encargada de manipular y estructurar los datos de la aplicación Web. Para ello, se conecta con la base de datos tanto para hacer las consultas solicitadas por la capa anterior, así como para publicar los datos enviados desde la API. En esta capa se modelan las entidades de Alarmas, Usuarios, etc.

La capa *Base de Datos* representa el conjunto de los datos almacenados con una determinada organización para que el sistema funcione correctamente. Para implementar esta base de datos se emplea *PostgreSQL*.

3.1.2 Modelo basado en casos de uso

El diagrama de casos de uso especifica las funciones o acciones disponibles en el sistema a desarrollar, es decir, lo que el sistema debe hacer.
En este diagrama se incluyen tanto los casos de uso como los actores involucrados en cada uno. Los actores pueden ser entidades humanas o no-humanas que se encuentren fuera del sistema. Este diagrama no muestra los elementos concretos del sistema sino su comportamiento, captando así sus requisitos funcionales.

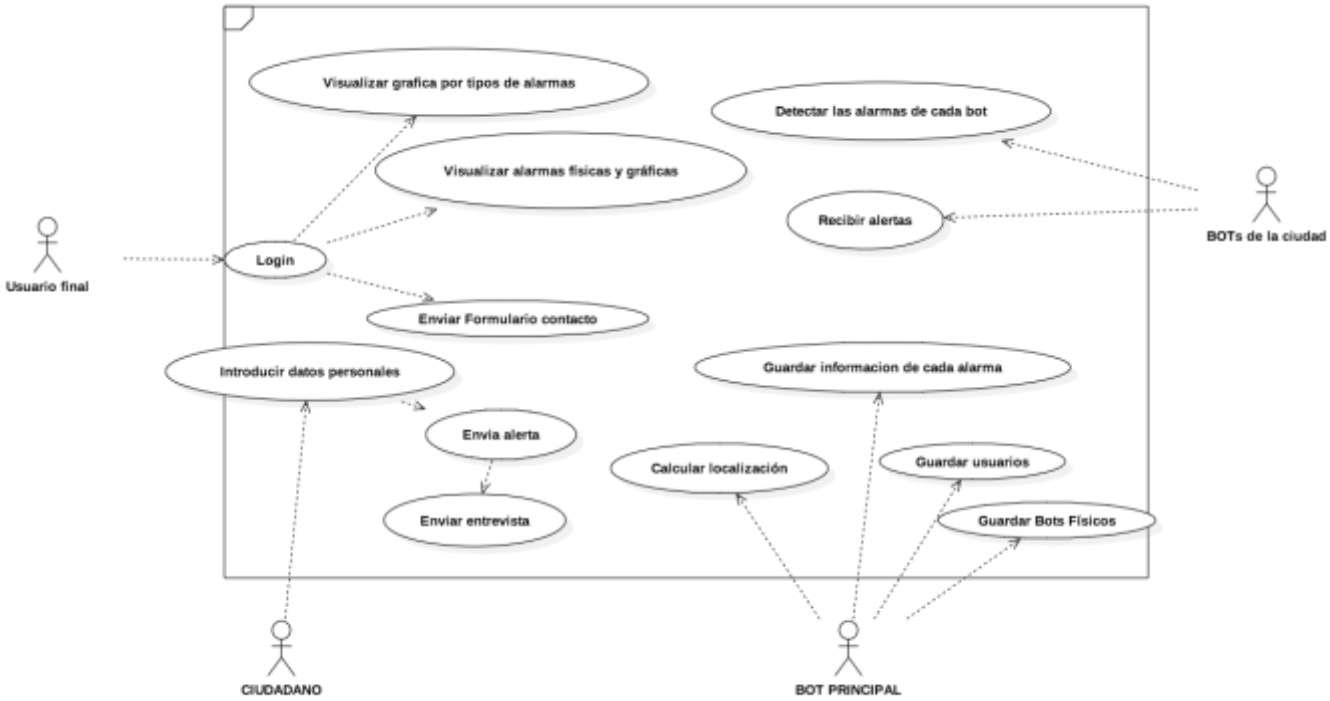

Figura 3: Diagrama de Casos de Uso

De estos casos de uso, se describen a continuación los más relevantes:

Nombre caso de uso	Login	
Descripción	Permite a los usuarios iniciar sesión en el sistema	
Actores	Usuarios finales que consumen el servicio web de visualización de datos	
Precondición	El solicitante debe encontrarse previamente registrado por el administrador	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario se dirige a la vista login de la aplicación web
	2	El sistema solicita el nombre de usuario y la contraseña
	3	El sistema comprueba si el usuario existe
Flujo Alternativo	**Paso**	**Acción**
	3	Los datos introducidos no coinciden con ningún usuario registrado y se informa del error
Postcondición	El usuario puede consumir los servicios	

Tabla 4: Caso de uso Login

Nombre caso de uso	Calcular localización	
Descripción	Calcula el Bot Físico (Alarma Física) más cercano a la reportada	
Actores	Bot Principal y Bots de la Ciudad	
Precondición	Algún ciudadano debe haber enviado una alarma al Bot	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario introduce una alarma con su localización
	2	El sistema consulta a la Base de Datos las localizaciones de los bots físicos
	3	El sistema implementa el algoritmo de localización
Flujo Alternativo	**Paso**	**Acción**
	3	La ubicación introducida por el usuario se encuentra a una distancia superior a la establecida
Postcondición	Se activa un estado del bot más cercano	

Tabla 5: Caso de uso Calcular localización

Nombre caso de uso	Visualización gráfica por tipos de alarmas	
Descripción	Permite a los usuarios visualizar una gráfica estadística	
Actores	Usuarios finales que consumen el servicio web de visualización de datos	
Precondición	El solicitante debe haber entrado al sistema y además, deben encontrarse alarmas publicadas en la Base de Datos	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario entra en la vista de visualización de la Aplicación Web
	2	El sistema consulta la BBDD para pintar la gráfica
Flujo Alternativo	**Paso**	**Acción**
	3	El sistema no encuentre alarmas registradas y no pueda dibujar la gráfica
Postcondición	El usuario puede consumir los servicios	

Tabla 6: Caso de Uso Visualización gráfica por tipos de alarmas

Nombre caso de uso	Guardar usuarios	
Descripción	Permite el registro de los usuarios del bot PopayanAmigo	
Actores	Bot Principal	
Precondición	El Bot debe haber enviado todos los datos de manera correcta	
Secuencia Normal	**Paso**	**Acción**
	1	El Bot envía los datos en formato JSON a una url determinada
Flujo Alternativo	**Paso**	**Acción**
	3	Se haya enviado algún dato en formato incorrecto o no se envíen todos los datos requeridos
Postcondición	El sistema almacena al usuario en la BBDD	

Tabla 7: Caso de uso Guardar usuarios

3.1.3 Modelo basado en componentes

En el diagrama basado en componentes se pueden distinguir dos grandes componentes que son los dos módulos incluidas en el proyecto desarrollado en Django (Ver *sección 3.1.7*).

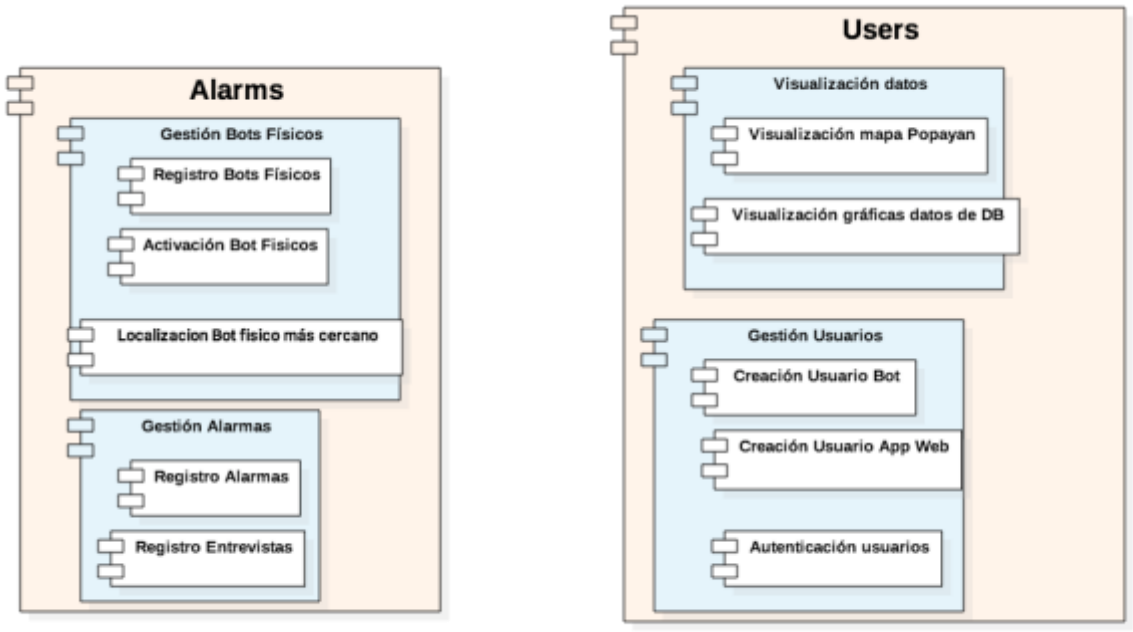

Figura 4: Diagrama de componentes

Componente Users:

Integra todos los casos de uso orientados al manejo de usuarios, incluyendo la lógica de negocio, la interfaz gráfica, las transacciones con la base de datos y demás tareas alrededor de este módulo.
Maneja todo lo relacionado con los usuarios del bot *PopayanAmigo* y de la Aplicación Web, además en él se encuentran los componentes encargados de la visualización.

- Gestión de Usuarios: Relacionado con los casos de uso: Login, introducir datos personales y guardar usuarios. Es donde se realizan las operaciones sobre la base de datos de usuarios, por lo que implementa las tareas tanto para los usuarios del bot *PopayanAmigo* como para los usuarios de la Aplicación Web.

- Visualización Datos: Se relaciona con los casos de uso: Visualizar grafica por tipo de alarma y visualizar alarmas físicas y gráficas, incluye la lógica de negocio relacionada con la interfaz gráfica.

Componente Alarms:

Integra los casos de uso orientados al manejo de alarmas y de bots físicos, incluyendo la lógica de negocio y las transacciones con la base de datos orientadas a las alarmas.

- Gestión Alarmas: Se relaciona con: Enviar Alerta, Enviar Entrevista, Guardar Información de cada alarma. Y es donde se realizan las operaciones sobre la base de datos de alarmas.
- Gestión Bots Físicos: Se relaciona con: Calcular localización, Guardar Bots Físicos, Guardar información de cada alarma y detectar las alarmas de cada Bot. Realiza operaciones en la base de datos de Bots y alarmas, e implementa el algoritmo de localización.

3.1.4 Diagrama de comunicación

En esta vista, se describen los protocolos de comunicación empleados en el sistema implementado en este TFG.

De acuerdo a la *Figura 4*, la API desarrollada en Django se comunica con la Base de Datos configurada mediante consultas SQL (Structured Query Language), que es un lenguaje de consulta estructurado. A diferencia de muchos lenguajes informáticos no es difícil de leer y entender. Es un estándar internacional para las consultas a bases de datos, reconocido por los organismos de normalización como ISO [11].

Por otro lado, el sistema externo reflejado en los Bots se comunica con la API mediante el protocolo de comunicación HTTP (*hypertext tranfer protocol*). Este proceso de comunicación entre cliente y servidor consiste en parejas de solicitudes y respuestas HttpRequest y HttpResponse. Los métodos de petición que emplea son GET, POST, PUT, PATCH y DELETE. También se emplea HTTP como protocolo de comunicación entre el usuario de la Aplicación Web y la Interfaz Gráfica, y entre ésta y la Lógica de Negocio.
El resto de comunicaciones entre los módulos son llevadas a cabo internamente por Django.

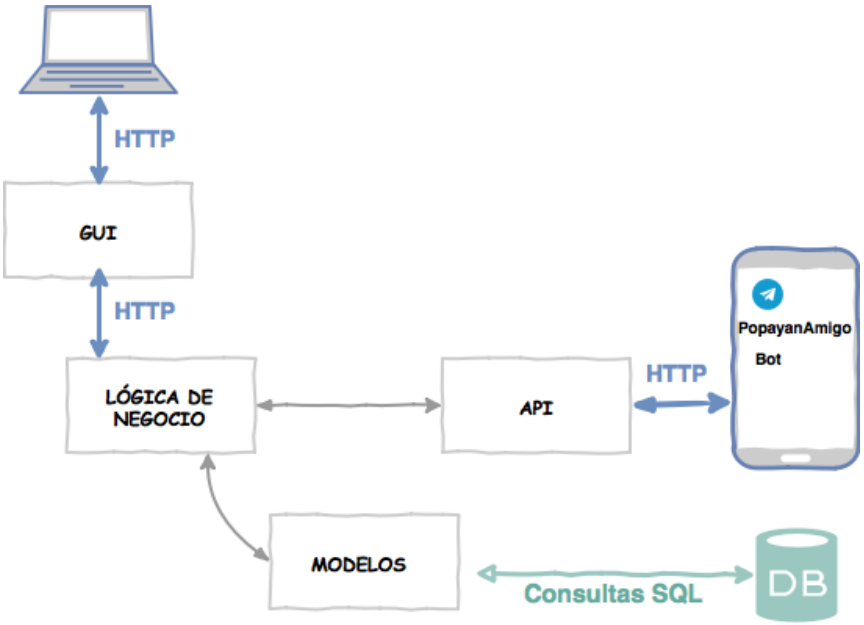

Figura 5: Diagrama de Comunicación

3.1.5 Diagrama de datos

Se desarrolla en base al Modelo Entidad/Relación, que es una técnica de análisis basada en la identificación de las entidades y de las relaciones que se dan entre ellas.
Se define *entidad* como objetos, reales o abstractos distinguibles de otros. Los *atributos* son las características de las entidades (asociadas a 1 o varias entidades). Y, las *relaciones* son las conexiones semánticas entre entidades.

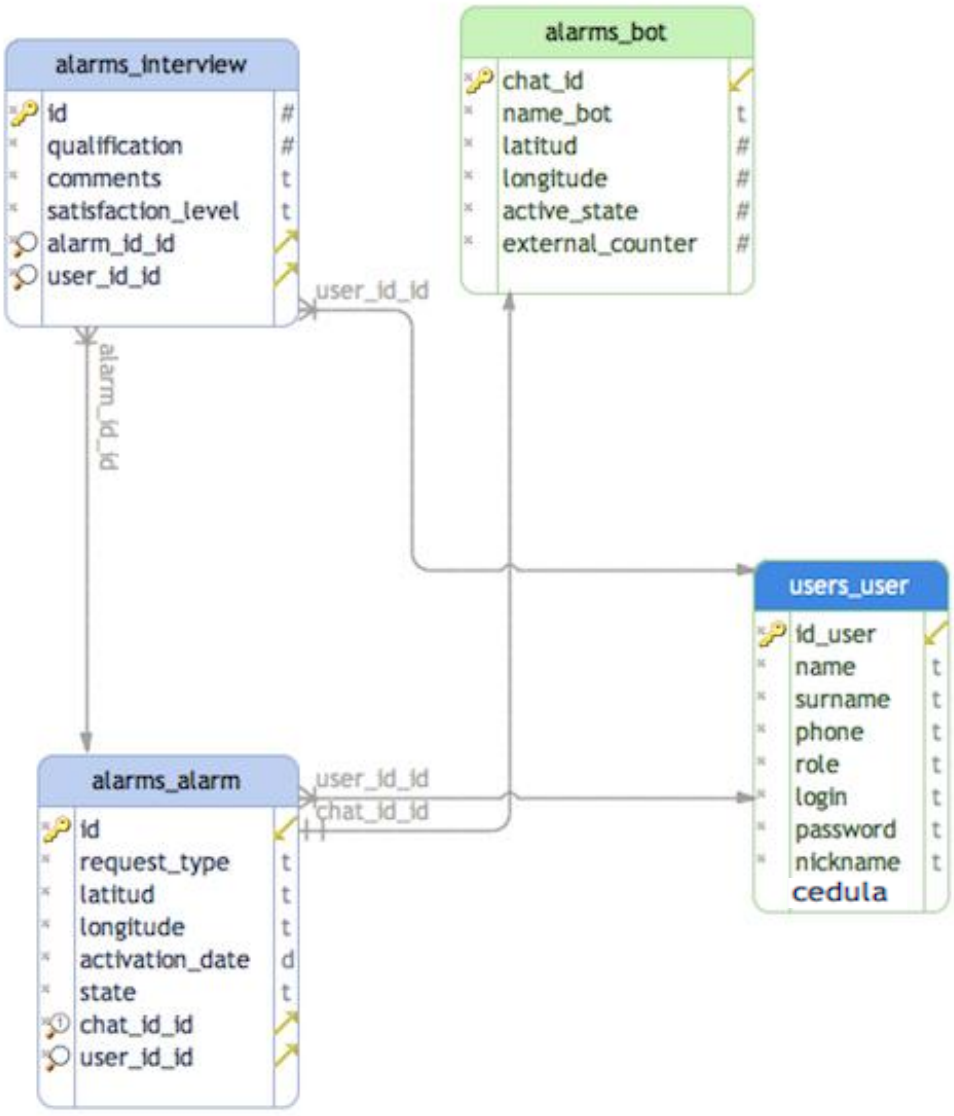

Figura 6: Diagrama de Datos

3.1.6 Diccionario de Datos

Es una descripción escrita de los datos almacenados en la base de datos (y de las restricciones de integridad que sean aplicables). Está formado por 4 entidades: User, Alarm, Interview y Bot.

User: Entidad que representa los usuarios conectados con el bot *PopayanAmigo* a través de Telegram y los usuarios del sitio web.

Atributos	Descripción
id_user	Identificador de la entidad User. Clave primaria (pk)
name	Nombre del usuario
surname	Apellido del Usuario
phone	Teléfono del Usuario
role	Rol (Administrador, Alcaldía o Policía)
login	Nombre de Usuario para acceder al sitio web
password	Contraseña para acceder al sitio web
nickname	Nickname del usuario en la aplicación Telegram
cedula	Cedula del usuario

Tabla 8: Entidad User

Alarm: Entidad que representa las alarmas publicadas por los usuarios mediante conversaciones con el bot *PopayanAmigo*.

Atributos	Descripción
id	Identificador de la entidad Alarm. Clave primaria (pk)
request_type	Tipo de alarma (Robo, Riña, Accidente o Incendio
latitud	Latitud de la ubicación de la alarma
longitude	Longitud de la ubicación de la alarma
activation_date	Fecha en la que se inserta (o publica) la alarma
chat_id	Identificador del bot físico desde el que se activa o se localiza como más cercano. Clave Foránea (fk)
user_id	Identificador del usuario que publica la alarma (fk)

Tabla 9: Entidad Alarm

Bot: Representa el conjunto de Bots Físicos distribuidos por la ciudad de Popayán.

Atributos	Descripción
chat_id	Identificador de la entidad Bot Físico. Clave primaria (pk)
name_bot	Nombre del bot
latitud	Latitud de la ubicación del Bot Físico en Popayán
longitude	Longitud de la ubicación del Bot Físico en Popayán
active_state	Variable que indica si se publicó una alarma cercana al bot, y es necesario activar la alarma física (0 ó 1)
external_counter	Contador que indica si la alarma se publicó en interacción usuario individual-bot PopayanAmigo

Tabla 10: Entidad Bot

Interview: Representa las entrevistas realizadas a los usuarios tras publicar una alarma.

Atributos	Descripción
id	Identificador de la entidad Interview. Clave primaria (pk)
qualification	Calificación (1-5) del servicio recibido
comments	Comentarios sobre el servicio recibido
satisfaction	Grado de satisfacción
alarm_id	Identificador de la alarma sobre la que se realiza la encuesta. Clave Foránea (fk)
user_id	Identificador del usuario al que se realiza la encuesta. Clave Foránea (fk)

Tabla 11: Entidad Interview

3.1.7 Estructura del código fuente

La disposición de archivos del código fuente del proyecto refleja la estructura básica que toda aplicación creada con Django sigue [12], además de incluir otros recursos necesarios para el correcto funcionamiento de la aplicación.

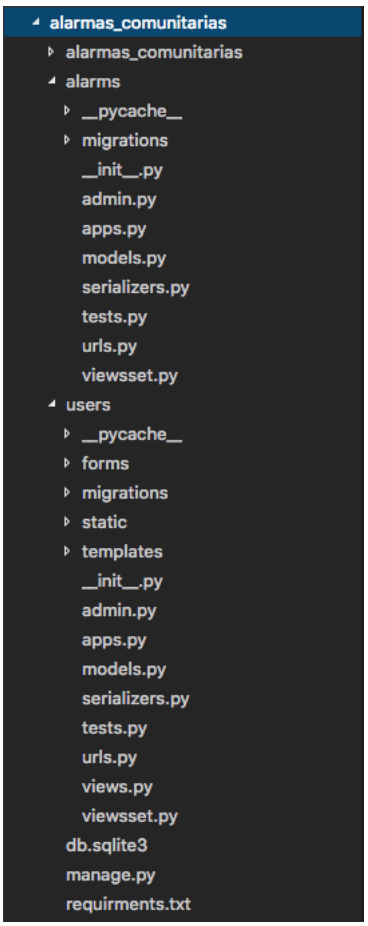

Figura 7: Estructura paquetes

De acuerdo a la estructura que aparece en la *Figura 6,* el sistema implementado en este trabajo consta de dos módulos: Alarms y Users.

- **alarmas_comunitarias/.**

  settings.py. Archivo que contiene toda la configuración del proyecto

  urls.py. Define las urls del sistema completo

  - **alarms/.** Módulo que contiene todo lo relacionado con las alarmas y los bots físicos.

    migrations/. Almacena un log de las modificaciones realizadas sobre la base de datos.

    admin.py Define el administrador de Django, que permite añadir, modificar y borrar elementos de las entidades

    models.py Define las entidades *Bot*, *Alarm* e *Interview* de la base de datos. Ver *sección 3.1.6.*

    urls.py Define las urls concretas de este módulo

    viewsets.py Define los puntos de comunicación con la API

  - **users/.** Módulo que contiene todo lo relacionado con los usuarios y el sitio web

    migrations/. Almacena un log de las modificaciones realizadas sobre la base de datos.

    forms/. Contiene los constructores de los formularios implementados en el sitio web.

    static/. Reúne los recursos de imágenes, estilo y los archivos js.

    templates/. Agrupa los archivos .html utilizados en la aplicación web

    admin.py Define el administrador de Django, que permite añadir, modificar y borrar elementos de las entidades

    models.py Define la entidad *User* de la base de datos. Ver *sección 3.1.6*

    urls.py Define las urls concretas de este módulo

    viewsets.py Define los puntos de comunicación con la API

    views.py Define la lógica asociada al sitio web, es decir a las templates

4 DESARROLLO

En este capítulo se va a detallar como se han desarrollado los componentes de cada capa del modelo de capas incluido en la sección 3.1.1.

4.1 CAPA GUI

La capa de interfaz gráfica corresponde al módulo de Templates de Django, por lo que está compuesta por todos los archivos (.html) y el componente static./ (ver estructura del código fuente en la *Figura 6* de la sección 3.1.7), que reúne los recursos de imágenes, estilos (.css) y los archivos js.

Las templates contienen partes estáticas de código HTML que constituyen la salida deseada, pero, a su vez, se mezclan con una sintaxis especial que describe como se insertará el contenido dinámico (variables enviadas desde la lógica). [13]

A continuación, se va a hablar de las tecnologías empleadas para desarrollar esta capa:

Bootstrap

Se ha empleado Bootstrap como *framework front end* que implementa el diseño web Responsive, como se ha introducido en la sección 2.2.1. Para el uso de esta tecnología se ha utilizado una plantilla ideal para desarrollar Aplicaciones Web de visualización de datos. Se han incluido los archivos de estilos de Bootstrap en la ya mencionada carpeta static./, y con la misma ha sido posible implementar los menús con su navegación, formularios, alertas, etc.

La plantilla se ha obtenido de: https://github.com/puikinsh/gentelella

Bootstrap se ha utilizado para la organización de contenidos en el template con el fin de proveer al sistema de la característica responsive, esto se consigue a través de la implementación del sistema grid de 12 columnas, para la división en columnas y filas de contenidos, y conseguir así un mejor entendimiento y acceso a la información por parte de los usuarios. Esto se aprecia en las figuras 7, 8 y 9, que muestran que el sitio web se representa en todos los formatos posibles.

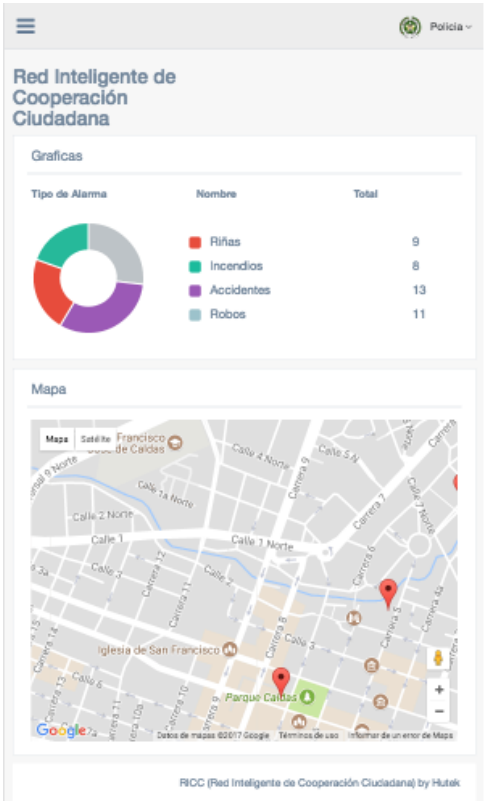

Figura 8: Página web formato móvil

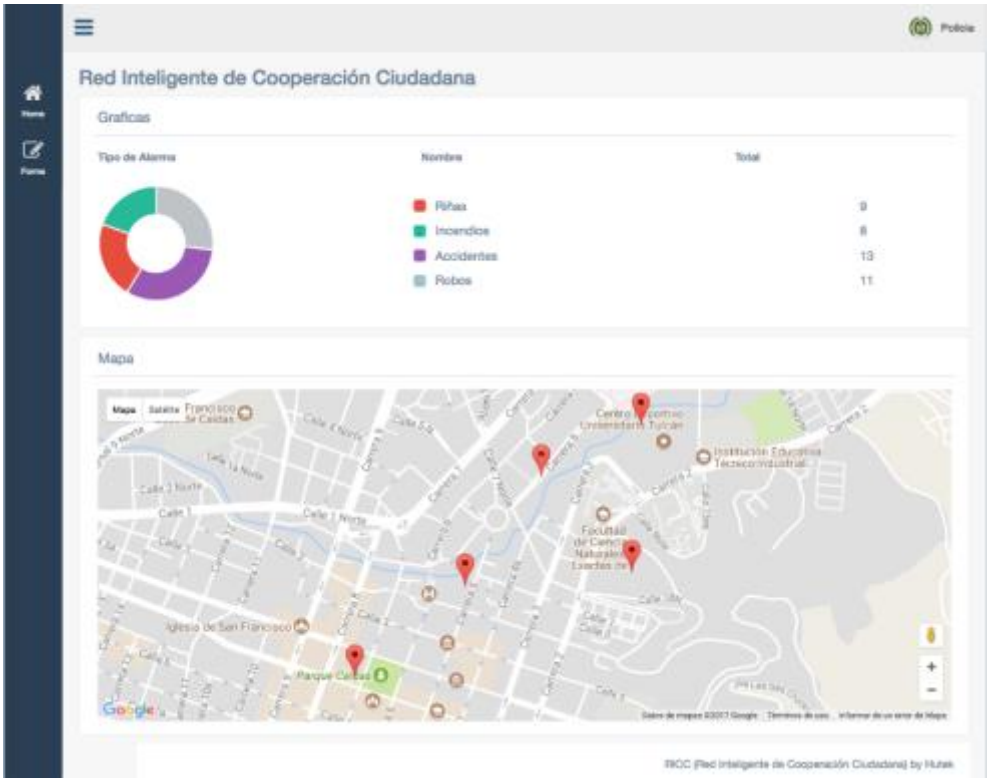

Figura 9: Página web formato Tablet

Figura 10: Página web formato PC

JavaScript

Para terminar de dar forma al sitio web se ha hecho uso de JavaScript. Las funcionalidades introducidas son:

Google Maps API

Se ha empleado la API de *GoogleMaps* para incluir los mapas en las diferentes vistas de la aplicación web [14]. Para ello se ha introducido el código JavaScript proporcionado por la documentación en el template *index.html*, y de esta forma insertar un mapa. Es necesario obtener una clave del sitio oficial de la API de GoogleMaps, e introducirla en el código. Los parámetros básicos del constructor son *center*, en el que se ha introducido el centro de la ciudad de Popayán (lat, long) y *zoom*, que indica el zoom con el que aparece el mapa cada vez que se recarga la página.

Así mismo, se han agregado marcadores en todos los puntos de la ciudad en los que haya un Bot Físico (o alarma física). Por lo que, se obtienen los datos desde la lógica que los extrae de la base de datos a través del modelo *Bot (Ver sección 3.1.6 y Tabla 8)*. Este proceso se realiza de forma automática, recorriendo un listado de bots mediante un bucle *for*, ya que se irán añadiendo más alarmas físicas a medida que el despliegue del sistema se vaya efectuando.

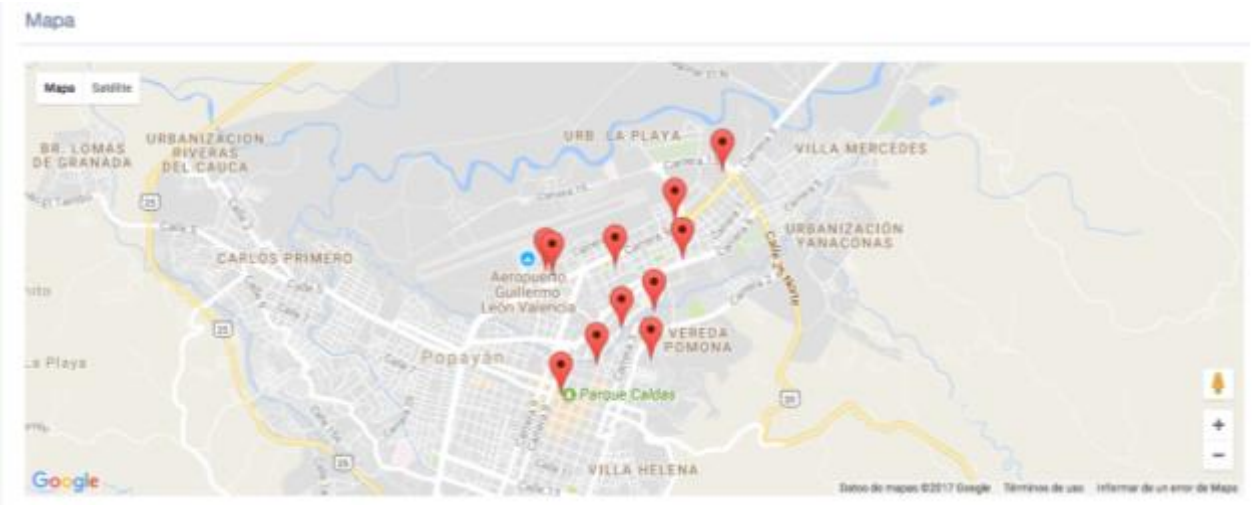

Figura 11: Mapa Popayán con marcadores

Comunicación con AJAX

Se ha empleado AJAX (Asynchronous JavaScript y XML) para obtener los datos de alarmas correspondientes a cada Bot Físico (o alarma física) al hacer click sobre cada marcador, y de esta forma poder pintar la gráfica. AJAX se define como una *"técnica de desarrollo web para crear aplicaciones interactivas y se trata de una tecnología asíncrona: los datos adicionales se solicitan al servidor y se cargan en segundo plano (background) sin interferir con el comportamiento de la página"* [15]

En *index.html* se ha implementado una función AJAX llamada cada vez que se hace click sobre un marcador. Envía el parámetro *chat_id* a la lógica, y así se puede filtrar y obtener el número total de alarmas por cada tipo asociadas a cada bot del modelo *Bot* de la base de datos. La información filtrada se devuelve en formato JSON al template, pudiendo pintar la gráfica correspondiente en cada caso.

Librería Chart.js

Esta librería se ha empleado para dibujar las diferentes gráficas que aparecen en el sitio web se ha utilizado la librería Chart.js. Esta se define como una *"librería simple pero flexible de gráficos JavaScript para diseñadores y desarrolladores"* [16].

De la misma se han obtenido las gráficas *dougnhut* y *line*, introduciendo su código JavaScript en scripts de los templates ya mencionados, y ajustando algunos parámetros. Los datos son enviados desde las views en la Lógica de Negocio. En el caso de los marcadores, al pinchar sobre cada uno se muestra una gráfica con las alarmas acontecidas en esa zona, como se observa en la *Figura 11*.

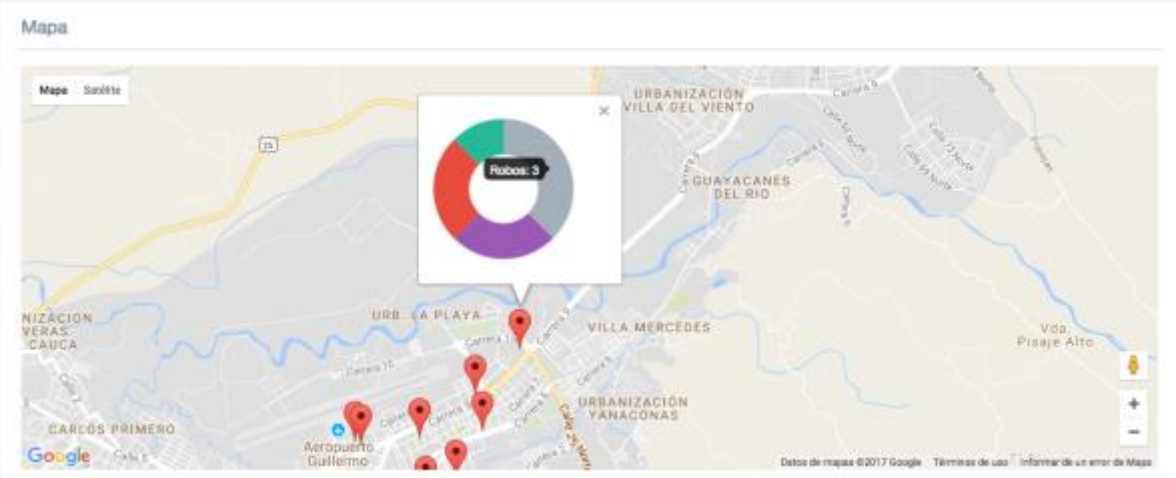

Figura 12: Gráfica de un marcador

4.2 CAPA LÓGICA DE NEGOCIO

En este bloque se detalla la lógica implementada en el trabajo. Se va a subdividir en dos secciones: una enfocada a la comunicación con la API, y otra enfocada a la comunicación interna del sistema.
Resulta interesante comentar que el proyecto de Django esta subdividido en dos módulos implementadas en este TFG: Alarms y Users.

4.2.1 Lógica de la API

En el proyecto de Django, estructurado previamente, esta parte se corresponde con los archivos viewset.py presentes en los dos módulos.

Se han creado 6 puntos de comunicación con la API, encargados de transmitir y recibir la información de la misma. Se resumen en la *Tabla 12*. La primera columna indica el nombre del servicio. El **Tipo de Método** se refiere al protocolo de comunicación http, que ya ha sido introducido en el Diagrama de Comunicación (Ver *sección 3.1.4*, *Figura 4*). Los métodos o directivas de petición determinan la acción que se aplica al recurso identificado por la *URL (Uniform Resource Locator)*.

Las directivas de petición empleadas en este proyecto son: GET, POST, PATCH.

GET: Cuando envía parámetros junto a la petición, los envía codificados en la *URL*.

POST: Cuando envía parámetros, van incluidos en el cuerpo de la petición.

PATCH: Se utiliza para actualizar recursos parciales, es decir, cuando sólo es necesario actualizar un campo del recurso y emplear PUT (directiva para actualizar recursos completos) consumiría más ancho de banda del necesario.

Los **Datos de Envío** son los parámetros enviados por el cliente a la API para recibir una respuesta. La **Respuesta (JSON)** son los datos generados por la API y se especifican las posibles respuestas para cada servicio. Por último, **URL** indica la dirección donde se consume el servicio, se especifican dos: la local (entorno de pruebas) y la del entorno de desarrollo.

Servicios	Tipo Método	Datos de Envío	Respuesta (JSON)	URL
Publicar Alarma	POST	request_type latitud longitude chat_id(opcional) user_id	response: “ok” response: “error”	Local: "http://127.0.0.1:8000/alarm_portal/alarms/"
Obtener lastAlarms	GET	horas	response: [{ id_user=21321, nombre="carlos", "peticion="robo", Telefono="234", Chat_id="2343", id_alarma="1231"}] response: error	Local: "http://127.0.0.1:8000/alarm_portal/last_alarms/"
envioEntrevista	POST	id_user qualification comments satisfaction alarm_id	response: "ok" response: "error"	Local: “htttp://127.0.0.1:8000/alarm_portal/interview”
estadoBot	GET	chat_id	response: [{active_state:"0 o 1"}] response: error	Local: “http://127.0.0.1:8000/alarm_portal/bot_state”

Actualizar Estado Bot	PATCH	active_state (0 ó 1) chat_id	response: {
			"chat_id": "XXX",
			"name_bot": "XX",
			"latitud": XXX,
			"longitude": XXX,
			"active_state": 0 ó 1,
			"external_counter": 0
			}
			response: "error"
				Local: “htttp://127.0.0.1:8000/alarm_portal/bot”

Registrar Usuario	POST	id_user, name, surname, nickname, phone (opc), role (opc), login (opc), passwd (opc) cedula (opc)	response: "ok" response: "error"	Local: “http://127.0.0.1:8000/user_portal/users”

Tabla 12: Servicios Web

Se establece qué parámetros de envío son necesarios, en formato JSON, y cuáles van a ser las respuestas para cada uno.

Algunos de los servicios desarrollados se basan en clases predefinidas por Django que permite una fácil codificación con el fin de recibir respuestas estándar de los mismos a partir del modelo de datos. Para servicios web específicos se realizó la lógica determinada de acuerdo con las necesidades del sistema y de los colaboradores.

Descripción de servicios

Publicar alarma

En este servicio se debe diferenciar de qué manera contacta el usuario con el bot *PopayanAmigo*:

- El usuario puede publicar alarmas desde un grupo de Telegram, asociado a una alarma física (o *Bot Físico*) de la ciudad de Popayán.

- El usuario puede establecer una conversación, o chat de Telegram, individual con el Bot.

En el primer caso la ubicación geográfica del grupo se encuentra previamente almacenada en la base de datos en la entidad *Bot*, incluida en el diccionario de datos de la sección 3.1.6; por lo que no es necesario llamar al algoritmo de localización, ya que se conoce el cuadrante o zona a la que el usuario se encuentra más próximo.

En cambio, en el segundo caso es necesario llamar a un algoritmo de localización que determine cuál es la alarma física más cercana, y en qué zona de la ciudad se encuentra el usuario, para así poder almacenarlo y activar dicha alarma física.

Para comprobar cuál es la forma en la que el usuario se está comunicando con el bot existe el parámetro *chat_id* (asociado a cada grupo o *Bot Físico*, de la entidad *Bot* del diccionario de datos). Si al intentar publicar una alarma desde la API se envía este parámetro, no es necesario llamar el algoritmo; en el caso de que se encuentre vacío, si se llamará.

En este servicio se realizó el diseño e implementación del algoritmo para detección de alarmas cercanas de acuerdo a la ubicación de usuarios del bot *PopayanAmigo*. El algoritmo es descrito en la siguiente sección.

Algoritmo de localización:

Descripción por Pseudocódigo:

1. Calcular distancia
2. Obtener elemento más cercano
3. Calcular distancia en kms
4. Si distancia < 300 m.
    a. Insertar alarma API asociada al bot identificado
    b. Cambiar estado de activación (active_state) a 1
5. Si distancia > 300 m.
    a. Devolver error: Usuario fuera del sistema RICC

Implementación en el sistema:

1. Calcular distancia: Esta parte del algoritmo se realiza a través de una consulta tipo SQL a la entidad *Bot* de la base de datos, teniendo la ubicación del usuario (*alarm_lar*, *alarm_long*) que informa de una alerta al bot *PopayanAmigo*. Y se ordena la tabla en función de la distancia a la que se encuentre cada elemento de la tabla de los parámetros enviados. Ese cálculo se hace mediante el teorema de Pitágoras.

```
locations = Bot.objects.raw('SELECT *, SQRT( POW((latitud - %s), 2) + POW ((longitude -  %s), 2)) AS distance FRO
M alarms_bot ORDER BY distance', [alarm_lat, alarm_long])
```

2. Obtener elemento más cercano: Se obtiene el primer elemento de dicha lista, que será el que se encuentre más cercano al usuario.

```
bot_cercano = Bot.objects.get(chat_id = locations[0].chat_id)
```

3. Calcular distancia en kms: Se calcula la distancia exacta en kms a la que se encuentra el usuario del bot identificado como más cercano, ya que es necesario establecer un máximo, para evitar que se puedan activar alarmas desde cualquier punto del mundo, muy alejado del sistema.

```
R = 6373.0
lat1 = math.radians(float(bot_cercano.latitud))
lon1 = math.radians(float(bot_cercano.longitude))
lat2 = math.radians(float(alarm_lat))
lon2 = math.radians(float(alarm_long))
dlon = lon2 - lon1
dlat = lat2 - lat1
a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2)**2
distance = R * c
```

4. Si distancia < 300 m.:
   Insertar alarma API asociada al bot identificado: Permite insertar dicha alarma en la API, asociándola al parámetro *chat_id* del bot identificado como más cercano, para poder realizar estadísticas y conocer en qué área se ha producido dicha alarma.

   Cambiar estado de activación (*active_state*) a 1: Se cambia el valor de *active_state* a 1 para que cuando los *Bots Físicos* lo estén consultando sepan que deben activarse.

5. Si distancia > 300m: Se devuelve como respuesta un error, indicando que el usuario se encuentra fuera de la Red Inteligente de Cooperación y Convivencia.

Obtener lastAlarms

El bot principal solicita a la API un listado de las últimas alarmas publicadas en un rango de tiempo, para poder realizarles la entrevista o encuesta sobre el servicio prestado.

En los datos de envío de este servicio se encuentra el parámetro *horas*, el cual es utilizado para consultar las alarmas generadas iniciando desde el momento de la consulta menos el número de horas especificado en el parámetro.
Este parámetro es enviado por el bot *PopayanAmigo* cuando consume el servicio, por lo que será escalable, es decir, se podrá modificar siempre que se desee sin necesidad de hacer cambios en el servidor.

Se realiza el filtrado en el modelo *Alarm* de la base de datos a partir del parámetro *activation_date (ver sección 3.1.6 y Tabla 9)*

Estado Bot

Los bots asociados a las alarmas físicas, que se encuentran distribuidos por la ciudad, solicitan a la nube cuál es su estado de activación (*active_state*, Ver entidad Bot en el diccionario de datos de la *sección 3.1.6*). Dichas consultas se establecen cada 5 segundos, y en ellas envían su *chat_id* y se les devuelve el parámetro *active_state*. Los bots realizan esta consulta con el propósito de saber si el parámetro ha cambiado, es decir, si algún usuario se encuentra cerca y ha enviado una alarma en un chat individual con Telegram.

Actualizar Estado Bot

Este servicio es consumido por todos los *Bots Físicos* distribuidos en Popayán. Se emplea para actualizar la variable *active_state* a 0 una vez se haya activado la alarma física con el fin de limpiar esta variable, ya que mediante el servicio web EstadoBot se está consultando cada determinado tiempo. En este recurso se emplea el método PATCH, ya que sólo es necesario actualizar un campo del recurso. Se envía el nuevo valor de *active_state* junto con *chat_id* para identificar al bot.

4.2.2 Lógica del sistema

En este apartado se va a exponer cómo se ha desarrollado la lógica encargada de generar la información visible a los usuarios del sistema e interactuar con los mismos.

En la estructura de Django, se corresponde con el archivo views.py (*sección 3.1.7*), donde está la lógica y la carpeta forms./ (*sección 3.1.7*) en la que se definen los formularios que se van a introducir en la aplicación web.

La interfaz gráfica del sitio web está formada por un total de 4 templates: login.html, index.html, forms.html y graphics.html. Para cada uno se define una función encargada de manejar su lógica. A continuación, se describen dichas funciones:

Función de login.html

La vista asociada a este template se encuentra compuesta por un formulario de inicio de sesión, creado en el archivo forms.py, contenido en el directorio forms./.
Para ello se crea la clase *LoginForm* que hereda de la de la clase Form de Django [17], y se generan dos campos para nombre de usuario y contraseña.

Los pasos asociados a esta vista son los siguientes:

1. Se comprueba que el método empleado en la petición HTTP sea POST

2. Se hace una referencia al formulario *LoginForm* y se invoca la función de validación del formulario con el fin de comprobar que los campos estén correctamente completados

3. Si el formulario es válido: obtiene los datos de la interfaz y consulta a la base de datos a través del modelo *User*, buscando que la combinación de usuario y contraseña introducida se encuentre en ella.

    - Si el resultado de la comprobación es negativo, redirige al usuario a la página de login de nuevo, mostrándole un mensaje de error:

Figura 13: Vista login "Usuario no encontrado"

- Si el resultado de la comprobación es positivo: almacena los datos del usuario (*id_user*) en la sesión del navegador y mediante la función *render()* de Django, redirige al usuario a la página *index.html*.

Función de index.html

Los pasos seguidos en la lógica asociada a esta vista son:

1. Comprobar que la sesión del navegador contenga el parámetro *id_user* entre sus datos, y de esta forma saber si se ha iniciado la sesión desde la vista anterior.
   En el caso de que no se encuentre, redirigir al usuario a la vista de *login.html* nuevamente.

2. Consultar a la base de datos y obtener los campos *name*, *surname* y *role* asociados al objeto *User*, que ha iniciado sesión.
   En función del rol, escoger que imagen se va a enviar como imagen de perfil.

3. Consultar a la base de datos y obtener el número total de alarmas en función de cada tipo (robo, riña, accidente o incendio).

4. Obtener un listado de los *Bots Físicos* mediante el modelo *Bot*, para poder insertar los marcadores en el mapa.

5. Enviar la información a *index.html*

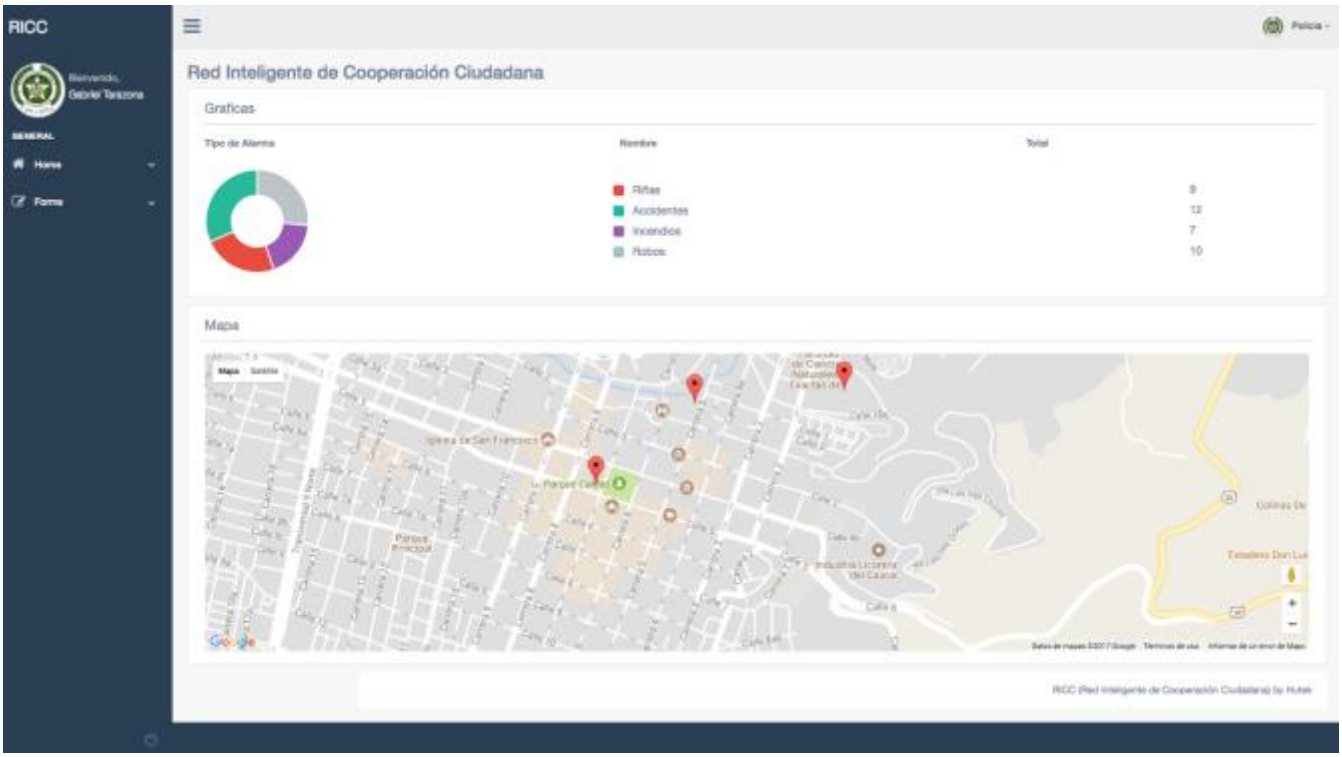

Figura 14: Vista Index

Por otra parte, se ha creado la función *filter_chatid(request)*, que se encarga de devolver la información solicitada a través de AJAX cada vez que se hace click sobre uno de los marcadores del mapa para dibujar las gráficas. El proceso seguido por esta función es:

- Obtener el parámetro *chat_id* enviado a través de AJAX desde la interfaz gráfica
- Buscar en la base de datos todas las alarmas asociadas a ese parámetro
- Obtener el nº total de las alarmas, en función de su tipo (robo, riña, accidente, incendio)
- Formar un diccionario con esos datos
- Devolver una respuesta de tipo JSON con dicho diccionario

Por otro lado, se ha creado la función *logout (request)*, que puede ser accedida desde cualquier lugar de la interfaz, se llama cada vez que se presiona la opción Logout en cualquiera de las vistas del sitio web. Esta función se encarga de limpiar la sesión, es decir, borrar los datos almacenados en ella y redirigir al usuario a *login.html.*

Función de form.html

En este caso se crea la clase *ContactForm*, que hereda de la clase Form y se generan los campos para asunto, nombre, email, teléfono y contenido del mensaje.

Los pasos asociados a esta vista son:

1. Si el tipo de petición HTTP es GET, que suele ser el caso de entrar en la página por primera vez, previo a rellenar el formulario:

   - Se repiten los pasos 1 y 2 de la función del *index.html*
   - Se envían los datos al template *form.html*

2. Si la directiva de petición HTTP es POST, caso en el que se ha completado el formulario de contacto y se ha pulsado enviar:

   - Se hace una referencia al formulario *ContactForm* y se invoca la función de validación del formulario con el fin de comprobar que los campos estén correctamente completados y los tipos de datos que correspondan (Por ejemplo, en el caso de teléfono que solo contenga números)
   - Obtener los datos de la interfaz cumplimentada por el usuario
   - Formar el mensaje con los datos del formulario (Nombre Completo, Email, Teléfono, Asunto y Contenido)
   - Enviar el mensaje a la dirección hutek.cauca@gmail.com, dirección de los colaboradores del proyecto encargados del soporte de la Aplicación Web.
   - Direccionar al usuario de nuevo a *form.html*, con los campos a insertar en blanco.

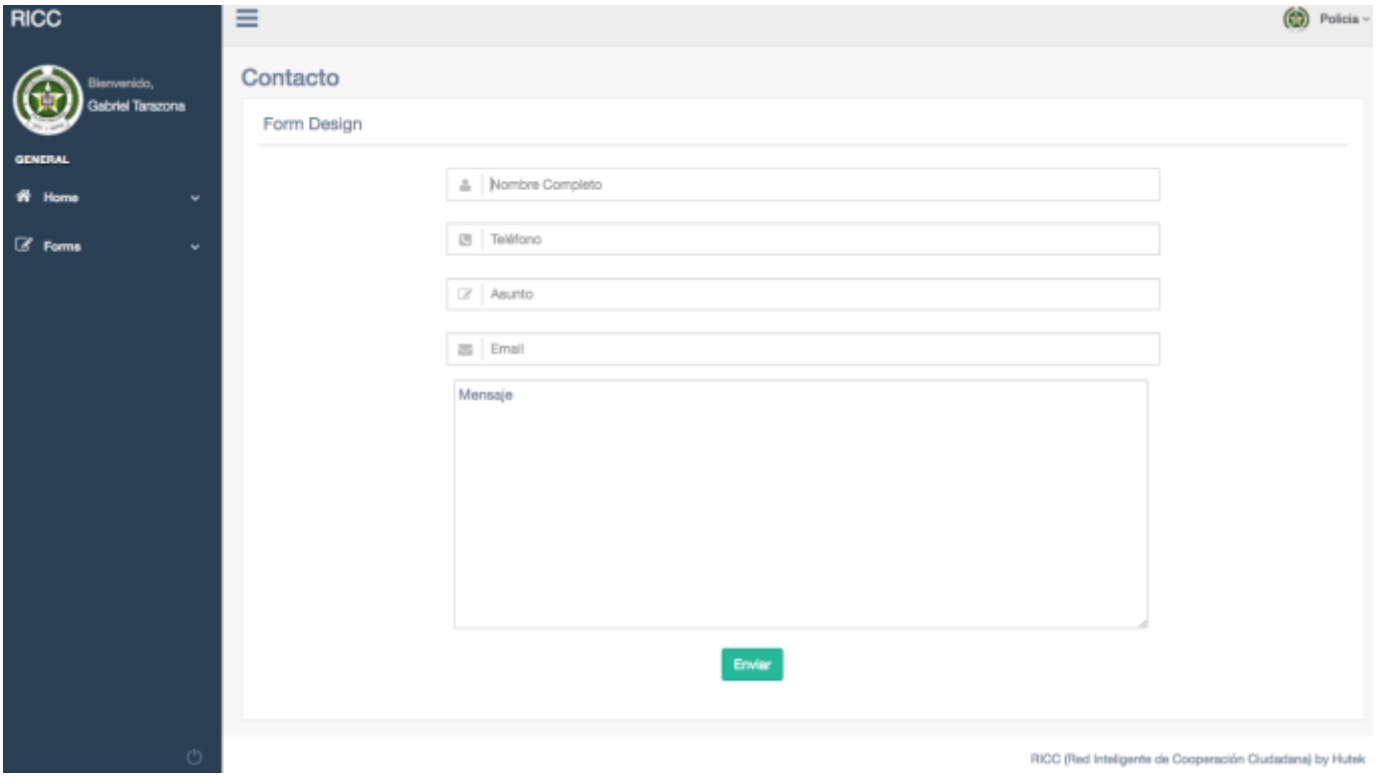

Figura 15: Vista Form

El envío de correo electrónico se hace mediante Django y éste utiliza SMTP (Simple Mail Transfer Protocolo), que se define como un protocolo de red utilizado para el intercambio de mensajes de correo electrónico [18]. Para ello es necesario configurar en el archivo settings.py del proyecto los siguientes parámetros:

```
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = ‘hutek.cauca@gmail.com’
EMAIL_HOST_PASSWORD = ********
```

En este caso se utiliza el servidor SMTP de Google para realizar el envío de correos.

Función de graphics.html

Los pasos seguidos en la lógica asociada a esta vista son:

- Pasos 1 y 2, seguidos en la lógica de index.html
- Consultar la tabla *Alarm* de la base de datos, en función del parámetro *activation_date*
- Obtener el número total de alarmas publicadas primero por año, y a continuación por meses
- Enviar la información al template graphics.html, para que pueda dibujar la gráfica de la *Figura 16*

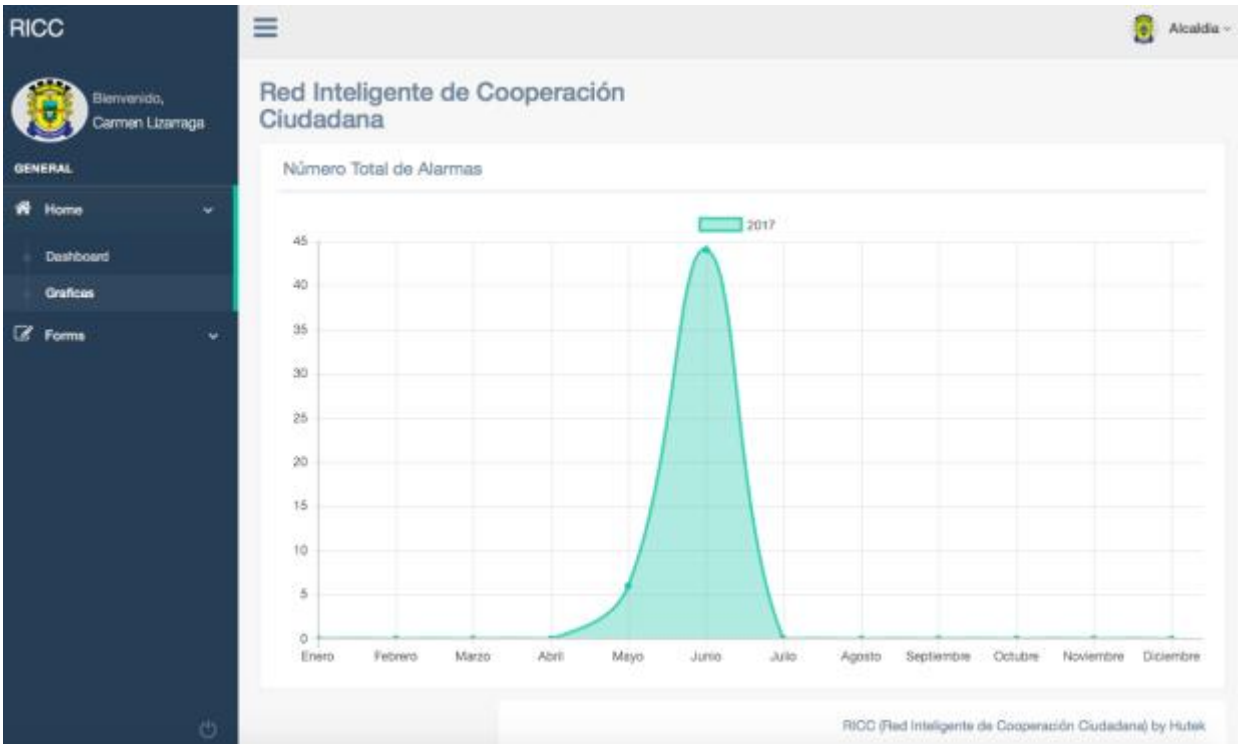

Figura 16: Vista graphics

4.3 CAPA MODELOS

En esta capa se engloban las partes de la aplicación encargadas de la gestión y manipulación de la información con la que la Aplicación Web opera: la base de datos y la lógica que necesita para su correcto funcionamiento.
Se corresponde con los archivos models.py contenidos en el módulo *Alarm* y el módulo *User (Ver sección 3.1.7)*.

El proceso seguido para implementar esta capa se describe a continuación:

1. Se hizo un diseño del diccionario de datos, mostrado en la *sección 3.1.6*. Un diccionario de datos se define como "*un repositorio centralizado de información sobre datos tales como significado, relación con otros datos, origen, uso y formato* " [19]. Es imprescindible realizarlo al principio del desarrollo del sistema, de esta manera se conocen que entidades y que parámetros son necesarios.

2. Se realizó un análisis de las relaciones entre las entidades definidas en el diccionario de datos, estableciendo que atributos de cada entidad son *Primary Key* o *Foreign Key*

3. Implementación de los modelos de cada entidad definida en el diccionario de datos.

4. Se hacen las migraciones en Django. Las migraciones son archivos que describen el conjunto de cambios que se han aplicado a la base de datos − creación, modificación o eliminación de tablas o atributos −. El proceso seguido es: implementar cambios en los modelos, crear las migraciones y por último migrar, que es la aplicación de dichas modificaciones a la base de datos, sincronizando así con los cambios realizados en el modelo.

5. Pruebas Base de Datos:
   Como primera fase del prototipo, en el entorno de desarrollo (local) se comenzó con *PostgreSql*, como se ha detallado en la *sección 2.2.2*, en este proyecto se utiliza una base de datos relacional, ya que es necesario una estructuración de los datos y posteriormente se realizó una migración al entorno de producción. En esa transición entre entorno de desarrollo y entorno de pruebas, se realizaron pruebas de consultas en el primero, utilizando el gestor de visualización PgAdmin [20], adecuado para las bases de datos *PostgreSql*.

5 PRUEBAS Y RESULTADOS

Se han desarrollado diversos tipos de pruebas sobre el sistema, con el fin de detectar posibles errores y validar el sistema desarrollado. En términos generales se consideran prioritarias las pruebas relativas a:

- El funcionamiento y la correcta respuesta de los servicios web implementados.
- Respuestas del sistema completo.
- Correcto funcionamiento del algoritmo de localización
- La visualización de los datos de una manera concisa e intuitiva, y la satisfacción de los diferentes actores que los visualizan.

Tras esta priorización se presentan a continuación las pruebas diseñadas e implementadas, según del ámbito de la aplicación en el que se han desarrollado.

5.1 PRUEBAS DE LA API

En ellas se busca probar la comunicación, respuesta, envió de datos y validación de los servicios web.
Para llevarla a cabo se utiliza Postman. Es una extensión del navegador Google Chrome o aplicación, que permite probar los servicios web de una manera sencilla. Para ello únicamente es necesario incluir la url, el método http (GET, POST, PATCH, etc) y los parámetros asociados a cada petición [21].

La metodología seguida ha sido probar cada uno de los servicios web descritos en la *sección 4.2.1* en base a la *tabla 12* que define los parámetros de envío y las respuestas para cada uno.

Para llevarlo a cabo se han enviado los parámetros requeridos para cada servicio web y se ha comprobado que la respuesta dada por la API se corresponda con la respuesta esperada, incluida en cada tabla.

A continuación, se presentan las pruebas realizadas para cada servicio web descrito en la *sección 4.2.1*.

Publicar Alarma

Nº Prueba	Parámetros enviados	Respuesta esperada	Resultado
1	"request_type": "Accidente", "latitud": "2.1443", "longitude":"-76.2344", "chat_id": "12", "user_id": "12345"}	{"request_type": "Accidente", "latitud": "2.1443", "longitude":"-76.2344", "chat_id": "12", "user_id": "12345"}	INCORRECTO: Debido a que ese user_id no se encontraba registrado en el sistema, por lo que no se le permite publicar alarmas
2	{"request_type": "Accidente", "latitud": "3.1443", "longitude":"-76.2344", "chat_id": "12", "user_id": "1234"}	{"Codigo": "Error: el usuario se encuentra fuera del rango de RICC"}	CORRECTO: Dado que la localización enviada en latitud y longitud se encuentra muy lejos se obtuvo esa respuesta

Obtener LastAlarms

Nº Prueba	Parámetros enviados	Respuesta esperada	Resultado
3	“horas”:”5”	[{"id": 82, "request_type": "Incendio", "latitud": "2.448349", "longitude":"-76.599678", "activation_date":"2017-06-29T16:24:37.091805", "chat_id": "1", "user_id": "1234" }]	CORRECTO: Dado que, en el momento de la consulta, en las últimas cinco horas solo se había publicado una alarma

Consultar EstadoBot

Nº Prueba	Parámetros enviados	Respuesta esperada	Resultado
3	“chat_id”:12	{"chat_id": "12", "active_state": 1}	CORRECTO: se envía el parámetro chat_id y se obtiene el valor de active_state asociado al mismo

Actualizar Estado Bot

Nº Prueba	Parámetros enviados	Respuesta esperada	Resultado

3	{
	"chat_id": "12",
	"active_state": 1,
	}
		{
		"chat_id": "12",
		"name_bot": "Terminal",
		"latitud": 2.451306,
		"longitude": -76.607692,
		"active_state": 1,
		"external_counter": 0
		}
			CORRECTO: se envía el parámetro chat_id y el nuevo valor del parámetro active_state asociado a ese chat_id y se modifica. Obteniendo como respuesta los nuevos datos de la entidad Bot

Publicar Entrevista

Nº Prueba	Parámetros enviados	Respuesta esperada	Resultado Prueba

4	{
	"qualification": 3,
	"comments":"Todo bien",
	"satisfaction_level":"alto",
	"user_id": "1234",
	"alarm_id": "39"
	}
		{
		"id": 2,
		"qualification": 3,
		"comments": "Todo bien",
		"satisfaction_level": "alto",
		"user_id": "1234",
		"alarm_id": 39
		}
			CORRECTO: La entrevista se publica correctamente

5	{"qualification": 5, "comments": "El servicio prestado ha sido perfecto", "satisfaction_level":"alto", "user_id": "1234", "alarm_id": "" }	{"qualification": 5, "comments": "El servicio prestado ha sido perfecto", "satisfaction_level":"alto", "user_id": "1234", "alarm_id": "" }	INCORRECTO: Debido a que no se puede publicar una entrevista sin referenciarla a alguna alarma ya existente en la base de datos

8 (servicio web Publicar usuario)	{
	"id_user": "12345",
	"name": "Test",
	"surname": "Test",
	"nickname": "test",
	"cedula": "13245",
	"phone": "12345",
	"role": "Alcaldia",
	"login": "test",
	"password": "test"
	}
		{
		"id_user": "12345",
		"name": "Test",
		"surname": "Test",
		"nickname": "test",
		"cedula": "13245",
		"phone": "12345",
		"role": "Alcaldia",
		"login": "test",
		"password": "test"
		}
			Correcto

Publicar Usuario

Nº Prueba	Parámetros enviados	Respuesta esperada	Resultado

3	{
	"id_user": "12345",
	"name": "Test",
	"surname": "Test",
	"nickname": "test",
	"cedula": "13245",
	"phone": "12345",
	"role": "Alcaldia",
	"login": "test",
	"password": "test"
	}
		{
		"id_user": "12345",
		"name": "Test",
		"surname": "Test",
		"nickname": "test",
		"cedula": "13245",
		"phone": "12345",
		"role": "Alcaldia",
		"login": "test",
		"password": "test"
		}
			CORRECTO:
			Dado que se publica un usuario y se obtiene como respuesta el POST realizado.

El sistema respondió como se esperaba. Sin embargo, los resultados incorrectos se debieron a que no se había tenido en cuenta que para publicar alarmas o entrevistas era necesario que existiera algún usuario o alarma respectivamente en la base de datos, y que se pudiera referenciar. Como se puede ver en el diagrama y diccionario de datos (sección 3.1.5 y 3.1.6). Tras estos resultados los errores fueron corregidos, y se especificó claramente que era necesario referenciar las alarmas y entrevistas publicadas.

5.2 PRUEBAS AL SISTEMA

En estas pruebas se va a medir el tiempo de respuesta de las diferentes vistas del sitio web, así como de las diferentes consultas a la API. De acuerdo con el Diccionario de IBM de Computación (que cita al vocabulario de la Organización Internacional de Normalización de Tecnologías de la Información vocabulario como fuente) el tiempo de respuesta se define como *"el tiempo transcurrido entre el final de una petición o demanda a un sistema informático y el comienzo de la respuesta correspondiente"*.

Según el libro *Usuability Engineering* de Jakob Nielsen del grupo Nielsen Norman, (consultora sobre la experiencia del usuario) hay 3 límites de tiempo principales a tener en cuenta al optimizar el rendimiento de un sitio web o aplicación web. Estos son:

- 0,1 segundos es el límite para que el usuario sienta que el sistema está reaccionando instantáneamente.
- 1.0 segundos es aproximadamente el límite para que el flujo de pensamiento del usuario permanezca ininterrumpido.
- 10 segundos es el límite para mantener la atención del usuario. Para tiempos de respuesta más largos, los usuarios desearán realizar otras tareas mientras esperan a que la página se cargue.

Siguiendo este estándar se han medido los tiempos de respuesta del sitio web implementado para la visualización en los dos entornos en los que se ha trabajado y desplegado el sistema (Ver *sección 2.2.3*)

URLS del sitio web	Tiempo de respuesta	**KB transferidos**
http://127.0.0.1:8000/user_portal/login	0.334 s	3.5 KB
http://127.0.0.1:8000/user_portal/index	2.8 s	21.2 KB
http://127.0.0.1:8000/user_portal/graphics	3.03 s	12.9 KB
http://127.0.0.1:8000/user_portal/form	0.637 s	11.4 KB

Tabla 13: Tiempos de respuesta Sitio Web entorno desarrollo

URLS del sitio web	Tiempo de respuesta
http://perfeccionamientoviewer.cloudapp.net:3000/user_portal/login	0.456 s
http://perfeccionamientoviewer.cloudapp.net:3000/user_portal/index	7.56 s
http://perfeccionamientoviewer.cloudapp.net:3000/user_portal/graphics	5.89 s
http://perfeccionamientoviewer.cloudapp.net:3000/form	4.52 s

Tabla 14: Tiempos de respuesta Sitio Web entorno de pruebas

En la *Tabla 13* se puede observar como ningún tiempo supera los 10 segundos de límite máximo establecido. Se puede observar como las vistas en las que no aparece ninguna gráfica tardan menos tiempo, sin llegar a 1 segundo.

En cambio, en la *Tabla 14* que refleja los tiempos de respuesta del sistema implementado en el entorno de pruebas, es decir, la máquina virtual o nube de Azure los tiempos de respuesta son muy superiores, casi triplicándose. Por lo que es evidente que es necesario agilizar el proceso, o desplegarlo en un entorno con mayores recursos.

A continuación, se han medido los tiempos de respuesta de las consultas realizadas a la API.

URLS de la API	Tiempo de respuesta	**KB transferidos**
http://127.0.0.1:8000/user_portal/users	1.1 s	13.1 KB
http://127.0.0.1:8000/alarm_portal/alarm	0.585 s	29 KB
http://127.0.0.1:8000/alarm_portal/bot	0.474 s	12.7 KB
http://127.0.0.1:8000/alarm_portal/interview	0.409 s	14.7 KB
http://127.0.0.1:8000/alarm_portal/last_alarms	0.417 s	12.7 KB
http://127.0.0.1:8000/alarm_portal/bot_state	0.353 s	9.8 KB

Tabla 15: Tiempos de respuesta API Entorno Desarrollo

URLS de la API	Tiempo de respuesta	**KB transferidos**
http://perfeccionamientoviewer.cloudapp.net:3000/user_portal/users	1.25 s	16.5 KB
http://perfeccionamientoviewer.cloudapp.net:3000/alarm_portal/alarm	1.43 s	57.8 KB
http://perfeccionamientoviewer.cloudapp.net:3000/alarm_portal/bot	0.743 s	10.6 KB
http://perfeccionamientoviewer.cloudapp.net:3000/alarm_portal/interview	0.828 s	24.9 KB
http://perfeccionamientoviewer.cloudapp.net:3000/alarm_portal/last_alarms	1.35 s	57.9 KB
http://perfeccionamientoviewer.cloudapp.net:3000/alarm_portal/bot_state	0.668 s	9.1 KB

Tabla 16: Tiempos de respuesta API Entorno Pruebas

En la *Tabla 15* y Tabla 16 se pueden observar los tiempos de respuesta de la API tanto para el entorno de desarrollo como para el entorno de pruebas, para las diferentes consultas. Como ha ocurrido en los tiempos de respuesta del sitio web los del entorno de pruebas (máquina virtual de Azure) son superiores, pero en este caso la diferencia no se considera relevante. En resumen, los tiempos son reducidos para ambos casos, por lo que, cuando los *Bots Físicos* se conecten a la API para consultar los diferentes servicios web obtendrán tiempos de respuesta rápidos, que son independientes del número de KB transferidos, como se puede observar en el caso de la segunda consulta de la Tabla 16 ya que transfiere 57,8 KB en 1.43 s. Además, se ha probado el tiempo de respuesta de la API a la hora de hacer un POST de una alarma y el resultado ha sido muy similar al mostrado en la tabla.

5.3 PRUEBAS A LA INTERFAZ

Esta prueba se ha realizado con el fin de comprobar la usabilidad, interacción y fácil entendimiento de la información mostrada en el sitio web de visualización. La validación de la interfaz gráfica se considera muy importante, ya que, en base a esta visualización de los datos, los diferentes actores (Policía, Alcaldía, Bomberos, etc.) podrán tomar decisiones e implementar políticas que reduzcan el alto índice de inseguridad en la ciudad de Popayán.

Para realizar esta prueba se consideran los siguientes criterios [22]:

- Eficacia: precisión con la que los usuarios alcanzan las metas específicas. Es decir, ¿los usuarios pueden hacer lo que necesitan en forma precisa?

- Eficiencia: ¿cuánto esfuerzo requiere que el usuario alcance su objetivo? Normalmente, la eficiencia suele medirse en términos del tiempo que les lleva a los usuarios realizar dichas tareas.

- Satisfacción: percepción de utilidad y agrado hacia el uso del producto. Es decir, ¿ cuál es la percepción del usuario frente a la facilidad de uso del producto?

A partir de estos criterios se diseñó la siguiente encuesta:

Pregunta 1: De 1 a 10 ¿qué tan fácil es identificar visualmente la información que brinda la plataforma?

Pregunta 2: De 1 a 10 Califique si los datos que observa en la plataforma ¿facilitan la toma de decisiones para aplicar políticas que reduzcan la inseguridad y las alertas?

Pregunta 3: De 1 a 10 ¿Cómo calificaría la rapidez de tomar decisiones en base a los datos visualizados?

Pregunta 4: De 1 a 10 ¿Cómo calificaría su experiencia de uso en el sitio web?

Se ha realizado la encuesta a un grupo de posibles usuarios, un total de 15, los resultados se presentan a continuación.

Pregunta	**Promedio**
1	7,54
2	8,18
3	8,09
4	8,18

Tabla 17: Resultado Promedio Encuesta

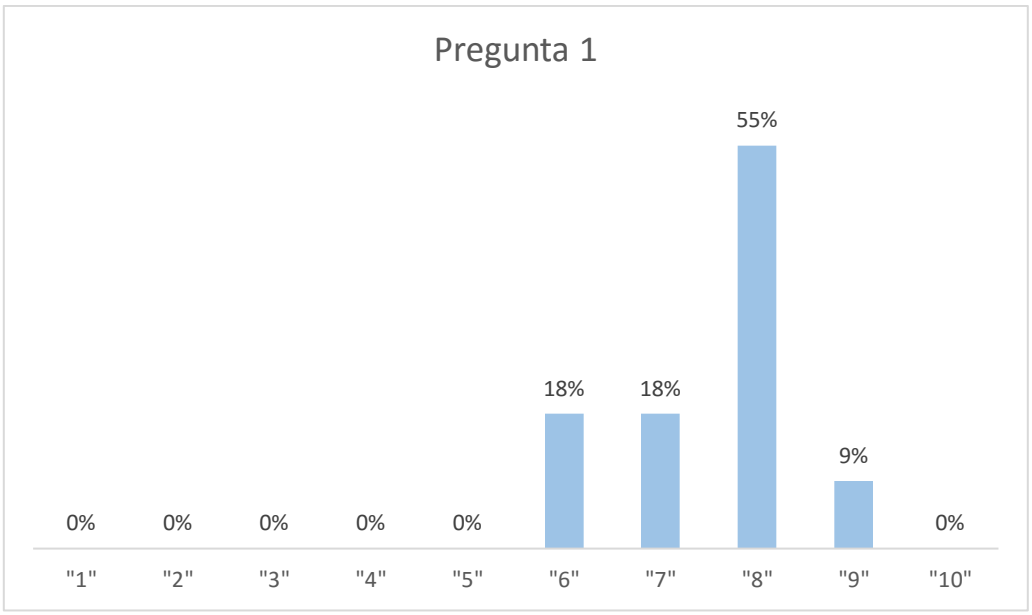

Figura 17: De 1 a 10 ¿qué tan fácil es identificar visualmente la información que brinda la plataforma?

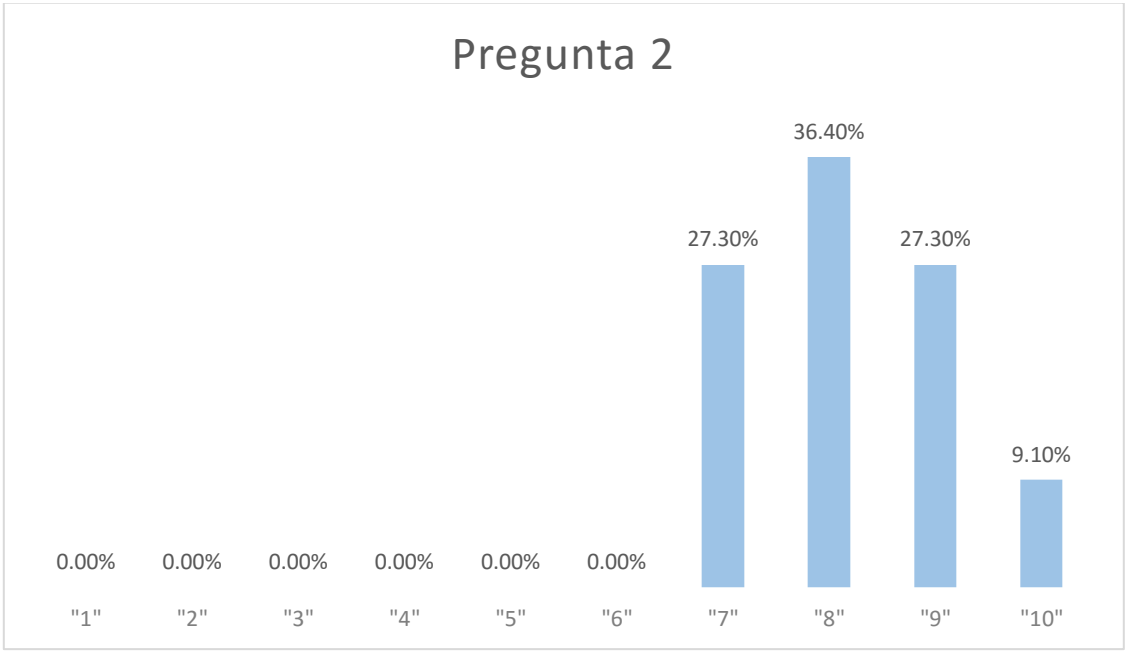

Figura 18: De 1 a 10 Califique si los datos que observa en la plataforma ¿facilitan la toma de decisiones para aplicar políticas que reduzcan la inseguridad y las alertas?

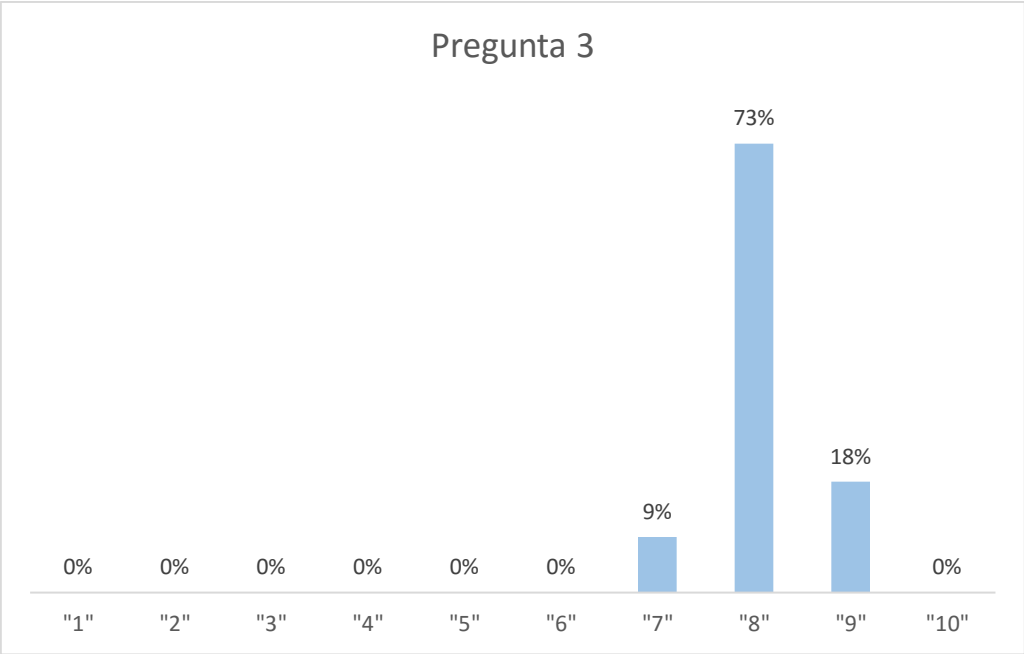

Figura 19: De 1 a 10 ¿Cómo calificaría la rapidez en tomar decisiones en base a los datos visualizados?

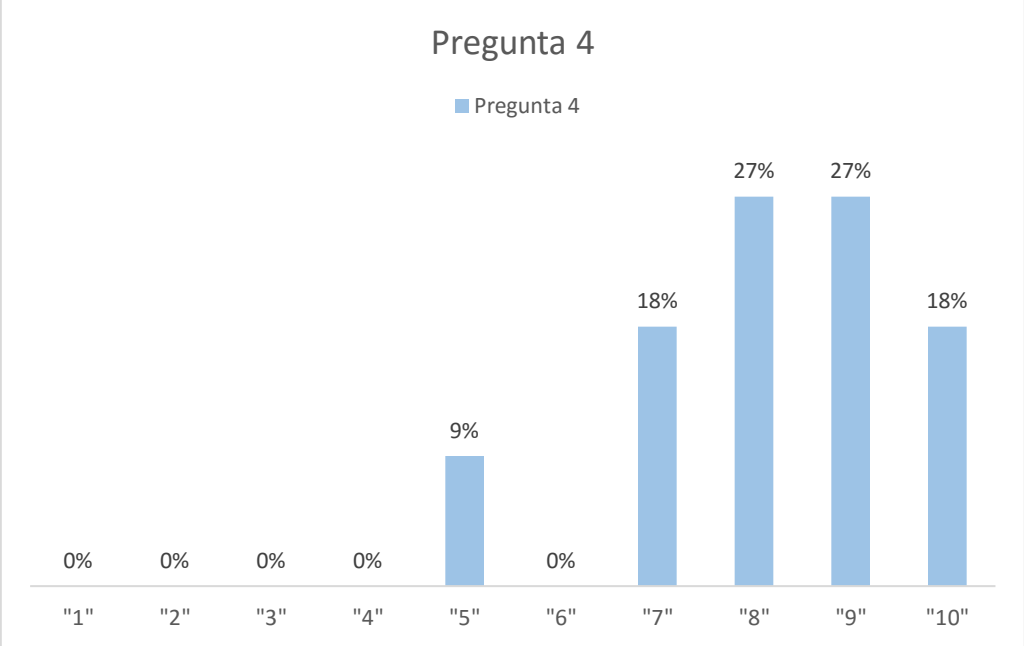

Figura 20: De 1 a 10 ¿Cómo calificaría su experiencia de uso en el sitio web?

Tras evaluar los resultados se obtienen las siguientes conclusiones:

- La pregunta con un peor resultado en promedio *(ver tabla 16)* se corresponde con la pregunta 1, por lo que la información, a pesar de ser sencilla de identificar visualmente supone algunas dificultades para el usuario.

- Los resultados relativos a la pregunta 2 señalan que el sitio web proporciona una alta facilidad para tomar decisiones, según los usuarios. Lo que se corresponde con el criterio de eficiencia definido previamente

- Los resultados obtenidos a partir de la pregunta 3 presentan que la toma de decisiones se puede encontrar rápidamente. Lo que guarda relación con el criterio de eficacia.

- La pregunta 4, obtiene un buen promedio, a pesar de obtener un 9% en la puntuación 5. En la Figura 21 se observa como tiene altos resultados.

En conclusión, tras la encuesta se ha obtenido una buena valoración de la plataforma diseñada e implementada. A pesar de ello, una posible línea de futuro de trabajo seria implementar gráficas más detalladas, o incluso emplear algún sistema de visualización de los que se encuentran actualmente en el mercado.

6 CONCLUSIONES Y LÍNEAS FUTURAS

Tras el desarrollo de este Trabajo de Fin de Grado y consecuentemente del sistema de almacenamiento, análisis y visualización de datos que complementa al sistema previamente desarrollado por el emprendimiento Hutek, se puede concluir que se ha cumplido el objetivo general definido al inicio del mismo: apoyar en la construcción de un sistema que contribuya a mejorar la seguridad de los ciudadanos de Popayán.

En el momento del inicio del desarrollo de este TFG, únicamente se recolectaban los datos generados por los usuarios. En el transcurso del trabajo se ha diseñado e implementado el almacenamiento de dichos datos, obteniendo unos resultados positivos en este aspecto, como se ha podido comprobar en las Pruebas a la API. Éstas presentaron algunos errores, que ya fueron solventados, por lo que en este aspecto el sistema implantado es todo un éxito.

Por otro lado, se ha desarrollado un algoritmo de localización, que ha supuesto un gran aporte al sistema de Hutek, ya que permite que las alarmas físicas dispuestas en Popayán se enciendan para alertar a los ciudadanos de cada barrio o cuadrante de que se está produciendo una alerta en su zona. Esto consigue que la ciudadanía de Popayán coopere como comunidad para reducir la inseguridad, y que la ciudad se transforme en ciudad inteligente, ya que se consigue gracias a la tecnología.

Por último, la plataforma de visualización ha obtenido buenos resultados en su valoración, y se espera que facilite en un alto grado la toma de decisiones a los actores principales que pueden reducir la inseguridad: Policía, Alcaldía, etc. Esto se consigue mostrándoles los datos de manera clara e intuitiva desde otra perspectiva, y no solo desde las denuncias interpuestas por los ciudadanos de Popayán.

Sin embargo, el sistema se puede mejorar considerablemente, para así obtener mejores resultados y que la inseguridad se reduzca aún más. Por lo que, dentro de las líneas de trabajo futuras, surge la idea utilizar potentes herramientas para la visualización de los datos. Una de ellas podría ser *Tableau*, posicionada actualmente como una de las herramientas líderes en el mercado de visualización de datos, que destaca por su accesibilidad para todo tipo de personas, ya que no requiere una familiaridad con la programación por parte de sus usuarios [23]. Además, es idónea cuando se trata de una enorme cantidad de datos, momento en el que no se encuentra este sistema actualmente, ya que se están realizando pruebas de despliegue. Por lo que, en un futuro, cuando sea utilizado por muchos usuarios y con lo cual, genere una gran cantidad de datos, se podría usar *Tableau* para la visualización.

Siguiendo esta idea, otra posible línea de trabajo futuro podría ser implementar el Big Data, con algoritmos de análisis de datos y predicción, que permitan en base a la alta cantidad de datos generados, predecir dónde y cuándo se van a producir alertas, lo que supondría un gran avance en la seguridad y reduciría los altos índices en los que se encuentran tanto Popayán como Colombia actualmente.

7 BIBLIOGRAFÍA

[1] L. Jaitman, «Los costos del crimen y de la violencia. Nuev evidencia y hallazgos en América Latina y el Caribe,» Nueva York, 2017.

[2] L. Chioda, Fin a la Violencia en América Latina, Washington DC: Grupo Banco Mundial, 2016.

[3] F. Cady, The Data Science, John Wiley & Sons, 2017.

[4] M. M. Muñoz, «Privacidad y procesado automático de datos personales mediante aplicaciones y bots,» *Dilemata,* nº 24, 2017.

[5] J. L. Roldan, G. C. y J. L. G. , «Los sistemas de inteligencia de negocio como soporte a los procesos de toma de decisiones en las organizaciones».

[6] FAQ:General, «Django Documentation,» [En línea]. Available: https://docs.djangoproject.com/en/1.11/faq/general/.

[7] «Merrian Webster,» [En línea]. Available: https://www.merriam-webster.com/dictionary/database.

[8] «Docs Microsoft,» [En línea]. Available: https://docs.microsoft.com/es-es/azure/documentdb/documentdb-nosql-vs-sqla .

[9] P. C. F. B. L. B. D. G. J. I. R. L. P. M. R. N. y J. S. , Documenting Software Arquitectures: Views and Beyond, vol. 2, Addison-Wesley Professional, 2011.

[10] C. M. University, «Software Engineering Insitute,» [En línea]. Available: http://www.sei.cmu.edu.

[11] v. y. s. Access SQL: conceptos básicos, «Microsoft Documents,» [En línea]. Available: https://support.office.com/es-es/article/Access-SQL-conceptos-básicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671.

[12] E. Django, «Documentación Django,» [En línea]. Available: https://docs.djangoproject.com/es/1.11/intro/tutorial01/.

[13] D. Templates, «Django,» [En línea]. Available: https://docs.djangoproject.com/es/1.9/topics/templates/.

[14] «Google Maps API,» [En línea]. Available: https://developers.google.com/maps/.

[15] C. J. &. L. S. S. S. Péndola Manrique, *Sistema de información para la planificación, control y reporte de servicios de confianza 'gasfiGuayas'.*

[16] «Chart.js,» [En línea]. Available: http://www.chartjs.org/.

[17] «Forms Django,» [En línea]. Available: https://docs.djangoproject.com/es/1.11/topics/forms/.

[18] «Django Email Docs,» [En línea]. Available: https://docs.djangoproject.com/en/1.11/topics/email/.

[19] IBM, Dictionary of Computing.

[20] «pgAdmin,» [En línea]. Available: https://www.pgadmin.org.

[21] «POSTMAN,» [En línea]. Available: https://www.getpostman.com.

[22] M. C. –. J. S. (Astrolabio), «Lineamientos y metodologías en Usabilidad para Gobierno en línea.».

[23] «Tableau,» [En línea]. Available: https://www.tableau.com/es-es.